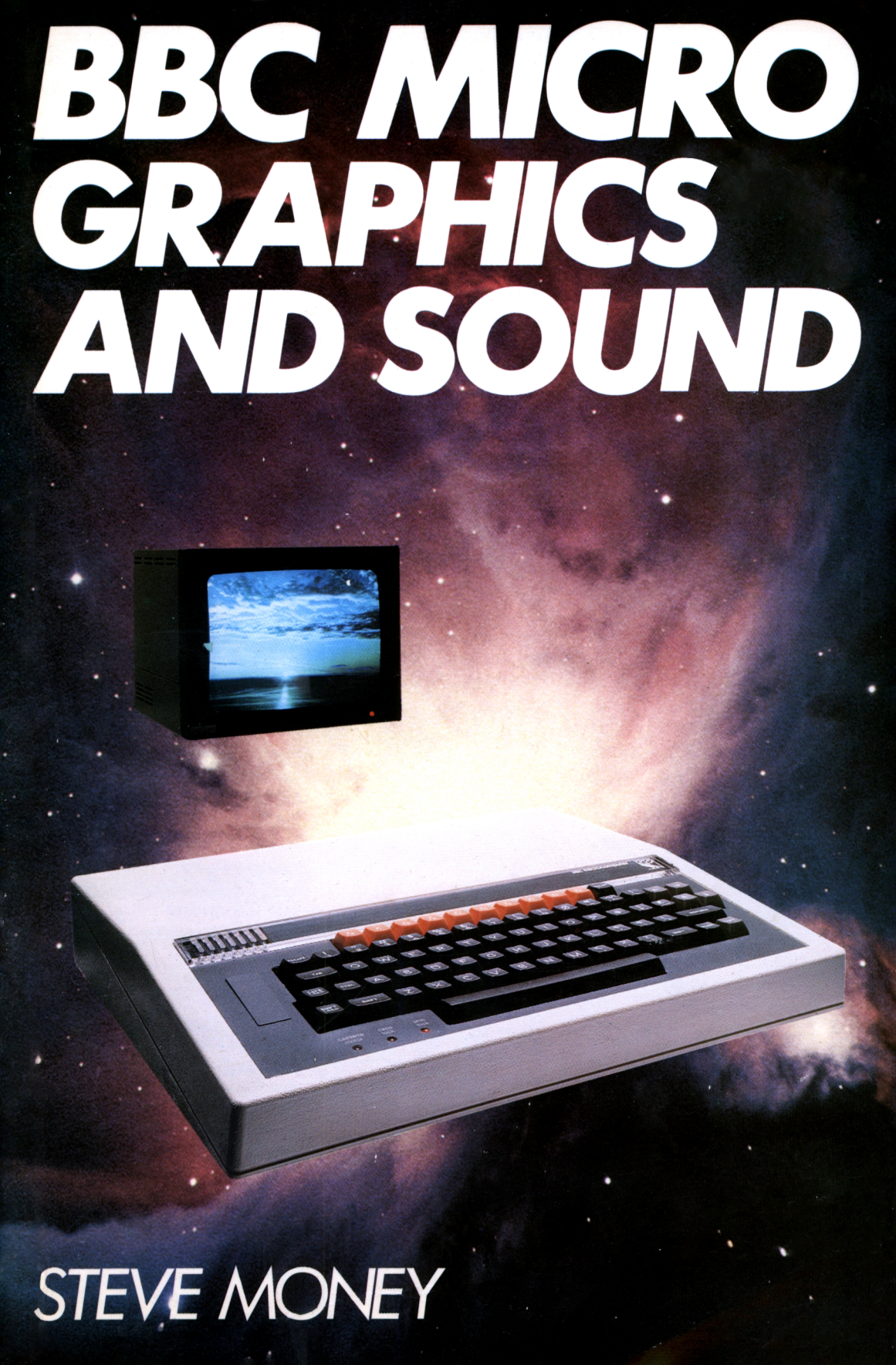
BBC MICRO GRAPHICS AND SOUND



STEVE MONEY

Other books of interest from Granada:

The Apple II

APPLE II PROGRAMMERS HANDBOOK
R. C. Vile
0 246 12027 4

ATARI

GET MORE FROM THE ATARI
Ian Sinclair
0 246 12149 1

THE ATARI BOOK OF GAMES
M. James, S. M. Gee and K. Ewbank
0 246 12277 3

The BBC Micro

INTRODUCING THE BBC MICRO
Ian Sinclair
0 246 12146 7

THE BBC MICRO – AN EXPERT GUIDE
Mike James
0 246 12014 2

21 GAMES FOR THE BBC MICRO
M. James, S. M. Gee and K. Ewbank
0 246 12103 3

BBC MICRO GRAPHICS AND SOUND
Steve Money
0 246 12156 4

DISCOVERING BBC MICRO MACHINE CODE
A. P. Stephenson
0 246 12160 2

ADVANCED PROGRAMMING FOR THE BBC MICRO AND ELECTRON
Mike James and S. M. Gee
0 246 12158 0

BBC MICRO AND ELECTRON MACHINE CODE – AN EXPERT GUIDE
A. P. Stephenson and D. J. Stephenson
0 246 12227 7

The Colour Genie

MASTERING THE COLOUR GENIE
Ian Sinclair
0 246 12190 4

The Commodore 64

COMMODORE 64 COMPUTING
Ian Sinclair
0 246 12030 4

THE COMMODORE 64 GAMES BOOK
Owen Bishop
0 246 12258 7

SOFTWARE 64 Practical Programs for the Commodore 64
Owen Bishop
0 246 12266 8

The Dragon 32

THE DRAGON 32 And How to Make The Most Of It
Ian Sinclair
0 246 12114 9

THE DRAGON 32 BOOK OF GAMES
M. James, S. M. Gee and K. Ewbank
0 246 12102 5

THE DRAGON PROGRAMMER
S. M. Gee
0 246 12133 5

DRAGON GRAPHICS AND SOUND
Steve Money
0 246 12147 5

THE DRAGON 32 How to Use and Program
Ian Sinclair
0 586 06103 7

The Electron

THE ELECTRON PROGRAMMER
S. M. Gee and Mike James
0 246 12340 0

21 GAMES FOR THE ELECTRON
Mike James, S. M. Gee and K. Ewbank
0 246 12344 3

BBC MICRO AND ELECTRON MACHINE CODE – AN EXPERT GUIDE
A. P. Stephenson and D. J. Stephenson
0 246 12227 7

Learning is Fun! 40 EDUCATIONAL GAMES FOR THE BBC MICRO AND ELECTRON
Vince Apps
0 246 12317 6

The IBM Personal Computer

THE IBM PERSONAL COMPUTER
James Aitken
0 246 12151 3

The Jupiter Ace

THE JUPITER ACE
Owen Bishop
0 246 12197 1

The Lynx

LYNX COMPUTING
Ian Sinclair
0 246 12131 9

The NewBrain

THE NEWBRAIN And How To Make The Most Of It
Francis Samish
0 246 12232 3

The ORIC-1

THE ORIC-1 And How To Get The Most From It
Ian Sinclair
0 246 12130 0

THE ORIC-1 BOOK OF GAMES
M. James, S. M. Gee and K. Ewbank
0 246 12155 6

Continued on inside back cover

BBC Micro Graphics and Sound

S. A. Money

GRANADA
London Toronto Sydney New York

Granada Technical Books
Granada Publishing Ltd
8 Grafton Street, London W1X 3LA

Copyright © 1983 by S. A. Money

British Library Cataloguing in Publication Data
Money, S. A.
BBC micro graphics and sound.
1. Computer graphics
2. BBC Microcomputer
I. Title
001.64'43 T385

ISBN 0-246-12156-4

First published in Great Britain 1983 by Granada Publishing
Reprinted 1983 (twice)

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed in Great Britain

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publishers.

Contents

Preface		iv
1	Introduction	1
2	Drawing and Plotting	15
3	Painting by Numbers	39
4	Graphs and Charts	52
5	More About Colour and Plot	71
6	The Teletext Mode	89
7	Making Things Move	104
8	The Third Dimension	115
9	Making Sounds	139
10	Making Music	156
Index		169

Preface

Among the more interesting developments in personal and home computer design in the past few years has been the introduction of high resolution colour graphics displays and versatile sound generation systems. Of the currently available home computers the BBC Micro probably offers the most comprehensive graphics and sound facilities.

The potential uses of the graphics and sound on the BBC Micro seem to be limitless and many hours can easily be spent in just trying out some of the many combinations that are possible. In this book the aim has been to explain the various facilities available on the BBC Micro in simple terms and to show how they might be used in practical applications.

Chapter 1 starts off by explaining the various techniques used for producing graphics and text displays and examines the various display modes available. This chapter concludes with a program which allows the computer to be used as a simple sketch pad.

The techniques of drawing various common shapes, such as circles, rectangles and polygons, are explored in Chapter 2 and a number of illustrative routines and short programs are included. The reader may then incorporate these routines and techniques into his own programs to allow virtually any shape to be drawn. It should be noted that all of the illustrations in this book were produced using a BBC Model B machine and an Epson MX80 printer.

Colour is an important factor in modern computer displays, particularly for video games applications, and in Chapter 3 the basic colour facilities of the BBC Micro are examined. Later in Chapter 5 some of the more advanced colour display features of the machine are explored. These include switching of the available colour set and the use of logical operations on colour commands to allow objects on the screen to move past one another without producing odd colour effects.

An important use for computer displays is for presentation of various types of graph or chart. In Chapter 4 the techniques involved in drawing bar charts, graphs and pie charts are dealt with.

One disadvantage of the BBC Micro is that for most of its display modes it tends to be very greedy on memory, with 10K and 20K of the available memory tied up solely for use by the display. Although this is compensated for by the excellent graphics facilities there are programs, such as adventure games, where a large amount of memory is required for the program itself. There is, however, one display mode which uses the same standard as broadcast teletext and requires only 1K of memory for the display. The various facilities available in this display mode are discussed in Chapter 6.

For games programs animation is often an important feature. Chapter 7 is devoted to the basic principles involved in producing animated images and also looks at the facilities available for producing special text symbols which may be useful in animated games programs.

The real world is of course three-dimensional and it is useful to be able to produce pseudo three-dimensional images on the computer display. The techniques producing three-axis graphics displays and perspective views are explained in Chapter 8 and a number of illustrative programs are provided. Real time animation of a perspective view normally requires the use of machine code programming to provide the processing speed and was felt to be beyond the scope of this present book, but a simple animation program in BASIC has been included to show the possibilities of this type of display.

With four channels the sound generator on the BBC Micro is extremely versatile and it would probably take a whole book and many years of experimenting to explore all of its possibilities. In Chapters 9 and 10 the basic operation of the sound generator is explored and the reader is shown how it can be used to produce various kinds of sound effects and music. The object here has been to show the reader how the sound and envelope facilities might be used.

Throughout this book the aim has been to explain in simple terms the various techniques needed for producing pictures and sounds from the BBC Micro. I have assumed that the reader is familiar with the general operation of the machine and its BASIC language. For those who find the *User's Manual* rather daunting I can recommend Ian Sinclair's book *Introducing the BBC Micro*. Each user will, of course, have his or her own particular requirements for producing displays or sounds and much of the fun of using a home computer is

the challenge of making it do what you want it to. I hope that in this book I have provided a useful set of tools to help the reader achieve that aim.

Finally I would like to thank Richard Miles of Granada publishing for his encouragement and help during the production of this book.

Steve Money

Chapter One
Introduction

One of the more attractive features of modern personal or home computers is their ability to produce graphics displays. When we talk about graphics this means the ability to produce drawings, pictures, graphs, charts and all sorts of other pretty patterns on the display screen.

All popular home computer systems can produce graphics of some sort and most of them provide full colour graphics. The quality of the displays we can produce depends upon the graphics *resolution*, which is a measure of how fine the details can be in a displayed picture. In general the higher the resolution the better the display, but usually the highest resolution displays will be limited to perhaps two or four colours.

Graphics displays can be very important when we are producing games programs and almost all modern computer games use graphics in one way or another. For arcade type games of the Invaders or Asteroids type, graphics are the key ingredient. In such a game not only do we draw a picture but we also make some, or perhaps all of, the objects displayed move around the screen. Here we are effectively using the computer to display a real-time moving picture of the game situation. Invaders move across the screen and we can shoot rockets or laser beams at them. With this type of game we are producing what amounts to a computerised version of the familiar cinema cartoon film.

Adventure type games can be greatly enhanced by actually showing a picture of the current location of the explorer, rather than just having a written description. It is also possible to show pictures of various objects or treasures that may be in the explorer's current location. It would also be possible to allow the explorer to examine a selected object in detail by expanding its size until it fills the screen. A variant of the adventure game is the maze, and here we can perhaps show a picture of what the explorer can see when he looks

along the path he is on. By using a pseudo three-dimensional display the side turnings from the present location may be shown.

Of course, although games are a pleasant way of passing the time we will also want to use the computer for more serious activities. In many applications we may want to produce various kinds of chart or graph to perhaps show how a business is progressing. Graphs and charts are also important in scientific applications where we want perhaps to display the results of a scientific experiment or the information gained from a poll or survey. Graphs and charts are generally much more effective at conveying information than a table of numbers.

Drawing on a computer using a high resolution graphics display is another application. We might perhaps produce technical drawings such as plans, electronic circuit diagrams, or book illustrations. The illustrations in this book were actually produced using a BBC Model B Microcomputer and an Epson MX80 printer. Many modern drawing offices use computer aided techniques for drawing and designing, which are provided by a set of programs called a *computer aided design* (CAD) package. We might use similar techniques to show the apparatus and demonstrate the results of a physics or chemistry experiment. This could be very useful where the actual equipment required would be very expensive. Here the computer simulates the actual experiment and the display might show the various stages in the progress of the experiment, giving readouts of perhaps temperature, pressure, etc.

Computer graphics can of course be used purely as an art form. Here we will be producing pretty patterns, handling colour and perhaps producing perspective views. Using techniques for displaying solid objects, it becomes possible to draw an object and then view it from different directions. The shape can then perhaps be modified and the result viewed until the desired effect is obtained.

The BBC Micro is perhaps one of the best of the currently available machines in terms of its graphics display capabilities. The model B in particular allows up to eight basic display colours on the screen at the same time and the high resolution graphics modes are excellent.

If you have bought the Model A, its capabilities are rather less impressive, but nevertheless it is capable of producing excellent graphics displays. The main difference in terms of graphics capability between the Model A and the Model B is simply the amount of memory available and in fact a Model A with its memory expanded to 32 kbytes can perform virtually as well as a Model B.

We shall be exploring the basic techniques of producing graphics in this book with particular reference to using the excellent capabilities of the BBC Micro. As a start in this direction let us first of all look at the way in which computers produce graphics displays and take an initial look at the various modes of operation available with the BBC machine.

The video display

All of the popular home computers, including the BBC Micro, use a television screen to provide a display. This may be the domestic television receiver or it could be a special television monitor designed for use as a computer *visual display unit* (VDU). At this stage it might be useful to take a look at the way in which the actual display is produced on the screen.

In a television system the image on the screen is actually traced out by a single moving dot which sweeps very rapidly across the screen in a series of horizontal lines. At the same time the dot moves rather more slowly down the screen so that each successive line falls just below the last one traced out on the screen. When the bottom right-hand corner of the screen is reached the dot moves back up to the left-hand top corner and repeats the whole scanning process again. This operation goes on continually with the dot sweeping down the screen every fiftieth of a second. The total number of scan lines is 625.

When we view this display our eyes are unable to see the dot moving because it travels so fast and the eye does not respond to things that change faster than about 20 times a second. As a result we see the whole picture as if it were present on the screen continuously. To reduce flickering the scan lines are interlaced. What happens is that on one sweep down the screen the dot traces out every other line, say the odd numbered lines, then on the next sweep it fills in the gaps by tracing out the even numbered lines. This gives a flicker rate of 50 per second which the eye cannot see, whereas tracing all of the lines once every $\frac{1}{25}$ second could produce a noticeable flicker which might be distracting to the viewer.

As the spot sweeps over the screen its brightness can be varied so that a pattern of light and dark dots appears, and these make up the picture that we see. A colour television has three sets of dots giving red, green or blue light, and by combining these we can produce colour displays.

Text displays

After you switch on your computer it will normally print up some sort of text message on the screen and before going on to look at graphics it might be as well if we looked at the way in which the computer displays printed text.

When the computer produces a printout of text on the screen it effectively divides up the screen area into an array of small rectangular spaces with say 40 spaces across the screen and 24 rows down the screen. Each of these rectangles is called a character or symbol space and in each of these spaces on the screen a single letter, number or other symbol may be displayed.

In the early days of computers, graphs and pictures were often produced by actually using text symbols to make up individual points in the picture. Some letters will appear brighter than others because of their shape, so by carefully choosing the letter placed at a point the image may be made light or dark. If the resultant page of text is viewed from a sufficient distance then it will show a picture, since the viewer's eyes will not be able to resolve the individual letters.

Graphics using symbols

Using text symbols is rather a crude way of getting pictures on the screen. An alternative method for producing pictures is to have a special set of graphics symbols in addition to the normal alphanumeric ones. These might contain vertical or horizontal lines running through the space at various positions, diagonal lines, curved lines and special symbols, such as musical note signs or playing card symbols. Try running the short program listed in Fig. 1.1 which will display some examples of this kind of symbol. By placing these special symbols in appropriate positions on the screen quite good drawings can be produced.

The BBC Micro does not have any of these special symbols in its normal character set but it is possible to produce them. There is a facility for creating your own special symbols on the BBC Micro, and we shall be exploring this feature in Chapter 7.

```
100 REM Special text symbols
110 MODE 2
120 VDU23,240,255,1,1,1,1,1,1,1
130 VDU23,241,0,0,0,0,15,8,8,8
140 VDU23,242,28,28,8,127,8,20,34,65
150 VDU23,243,8,8,8,8,15,8,8,8
160 VDU23,245,62,127,73,73,127,85,85,85
170 VDU23,246,8,28,62,127,62,28,8,0
180 VDU31,3,16
190 VDU 240,32,241,32,242,32,243
200 VDU31,3,20
210 VDU 244,32,245,32,246
220 END
```

Fig. 1.1. Program for displaying symbols.

Mosaic graphics

An alternative and more flexible approach to producing graphics is to have an extra set of symbols, where each text rectangle is divided up into a pattern of six small blocks arranged as two columns of three blocks each. If each block can be either black or white there are 64 different patterns of blocks that can be produced within the text symbol space. By carefully choosing the block patterns within the text spaces it is possible to produce a picture on the screen. Here we have a rather crude form of television picture. For a normal text display of 40 characters per line and 24 lines the picture is effectively built up from an array of dots 80 wide and 72 high. In fact this is rather better than Baird's original television system, which gave a resolution of about 40 by 30.

This technique for producing graphics is known as *mosaic graphics* and provides us with a low to medium resolution graphics capability. The technique is used for teletext and viewdata and is capable of producing quite effective pictures. On the BBC Micro there is a teletext display mode which uses the mosaic type of graphics display.

High resolution pixel graphics

If the text symbols on the screen are examined closely it will be seen that they are themselves built up from a pattern of dots. The BBC Micro uses a pattern which is eight dots wide and eight dots high for each text symbol space. The actual symbol is produced by selectively lighting the dots in the space as shown in Fig. 1.2. Normally the

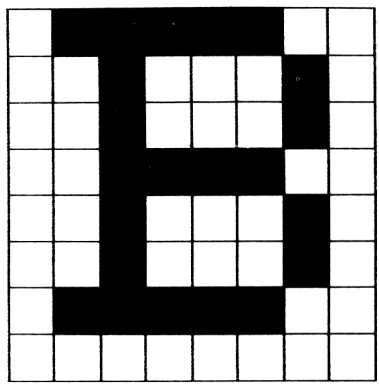


Fig. 1.2. The dot matrix used to produce text symbols.

patterns of dots for each symbol in the available set are stored in a special character generator memory and called up as a pattern when a particular symbol is to be displayed.

Suppose we could control the individual dots in every character space on the screen. Now for a 40 × 24 screen layout there would be 40 × 8 or 320 dots across the screen and 24 × 8 or 192 dots down the screen. The pictures produced would become much more detailed than can be achieved with mosaic graphics. In fact this is approaching the resolution available from a domestic video recorder.

The individual dots in each character space are generally referred to as *pixels* (picture elements) and this mode of graphics is called *pixel graphics* or high resolution graphics. On the BBC Model B the highest graphics resolution is actually 640 dots across the screen and 256 down the screen, giving a total of 160,000 individual pixels.

Display modes on the BBC Micro

The BBC Micro provides eight different display modes which are numbered as MODE0 to MODE7. Most of these display modes will permit both graphics and text to be displayed on the screen, but two modes (MODE3 and MODE6) are for text displays only.

The display modes available on the BBC Micro are:

Mode 0	Graphics 640 × 256	2 colours
Mode 1	Graphics 320 × 256	4 colours
Mode 2	Graphics 160 × 256	16 colours

Mode 3	Text	80 × 25	2 colours
Mode 4	Graphics	320 × 256	2 colours
Mode 5	Graphics	160 × 256	4 colours
Mode 6	Text only		2 colours
Mode 7	Teletext mode		

Note that a standard Model A BBC Micro with 16K memory will work only in modes 4, 5, 6 and 7, but an expanded Model A with 32K of memory will be able to use all display modes.

Let us examine each mode in a little more detail.

Mode 0 provides the highest graphics resolution with 640 dots across the screen and 256 rows down the screen. It is also possible to display text in this mode, with 80 text characters per row across and 32 rows of text down the screen.

Because of the high resolution of this mode it may be difficult to obtain a clear, sharp picture when using a domestic television as a display, but if a colour monitor is used with direct video drive then there should be no problems.

In this mode only two colours are permitted, one being the foreground or drawing colour and the other the background colour. Under normal conditions the background will be black and the picture or text will be white. We shall see later that it is in fact possible to choose any two colours from the available colour set for use in this mode instead of the normal black and white.

Mode 1 provides a lower graphics resolution of only 320 dots wide by 256 lines high. Text can also be displayed as 32 lines with 40 characters per line. Four colours are available in this mode, normally black, red, yellow and white. Any one colour may be chosen as the background and the graphics design or text may be displayed in the other three colours.

Mode 2 gives graphics of 256 × 192 or alternatively text appears as 32 rows of 20 characters per row. This mode provides the maximum range of colours with a total of sixteen selectable colours. In fact there are eight plain colours, which are black, red, green, yellow, blue, magenta, cyan and white. The other selectable colours are the same basic colours but they flash on and off.

Mode 3 allows only a text display with 80 characters per line and 25 lines. This mode permits only two colours, one for foreground and

one for background, and these are normally white and black, as for Mode 0.

Mode 4 is like Mode 1 with a graphics resolution of 320 × 256, but only two colours may be used, as in Mode 0. This mode allows text to be displayed in the format of 40 characters per row and 32 rows as for mode 1.

Mode 5 is like mode 2 with a graphics resolution of 160 × 256, but only four colours may be used. Text in this mode is 20 × 32 as for mode 2.

Mode 6 gives a text only display, with 40 characters per row and 25 rows. Only two colours can be displayed.

Mode 7 is a special mode called the *teletext* mode and is designed to be compatible with broadcast teletext services. Here both graphics and text may be used, but the graphics is of the mosaic type and is written on to the screen using the same method as for text display. We shall devote a special chapter to this mode later (Chapter 6) since it operates in a different way to the normal graphics modes. In mode 7 the effective graphics resolution is about 80 dots across the screen and 72 dots down the screen, giving relatively low resolution graphics. All eight colours and eight flashing colours can be used simultaneously in the teletext mode.

Memory requirements

One problem with high resolution graphics is that it requires a large amount of memory to store the data for the image on the screen, and the higher the resolution the more memory is needed. If we consider the mode 0 display with 160,000 pixels and if we use one bit of the computer's 8-bit word for each pixel, this means that a memory of 20,000 bytes is required just for the display. When it is considered that even the Model B has only 32,000 bytes, this leaves only 12,000 for programs and data.

For text displays most home computers use a dedicated character generator chip which provides the dot patterns for each symbol in response to a simple 8-bit character code. This means that only one word of memory is required for each symbol displayed. This technique is used on the BBC Micro for mode 7 and this requires a display memory of about 1000 bytes. In all other modes, however, the BBC Micro stores the actual pixel pattern in memory so that

either text or graphics use the same amount of memory. This has some advantages since it allows complete freedom in mixing text and graphics on the screen, but it does tie up a large part of the available computer memory for display generation.

The memory used for the screen display in each of the modes is shown below:

Mode 0	20K
Mode 1	20K
Mode 2	20K
Mode 3	20K
Mode 4	10K
Mode 5	10K
Mode 6	10K
Mode 7	1K

It will be seen that modes 0 to 3 need more than 16K of memory and will not work with a standard Model A machine, which has only 16K of memory available. An expanded Model A machine where the memory size has been increased to 32K, or a standard Model B machine will be able to run all display modes.

Text cursor and coordinates

When text is displayed a flashing bar appears on the screen and is called the text *cursor*. It shows the position on the screen where the next text symbol will be displayed. We can move this cursor around the screen by using the arrow keys at the right of the keyboard. When a new character is entered from the keyboard it takes the place of the cursor and the cursor moves on to the next symbol position in the row. If the cursor is at the end of a row it automatically moves down to the start of the next row.

When we are running a BASIC program it is possible to move the cursor about on the screen by using the TAB function which is inserted into a PRINT statement and takes the form

```
100 PRINT TAB (X,Y)
```

where X and Y are the coordinates of the position on the screen where we want to place the cursor. The value X is the number of spaces from the left-hand edge across the screen and Y is the number

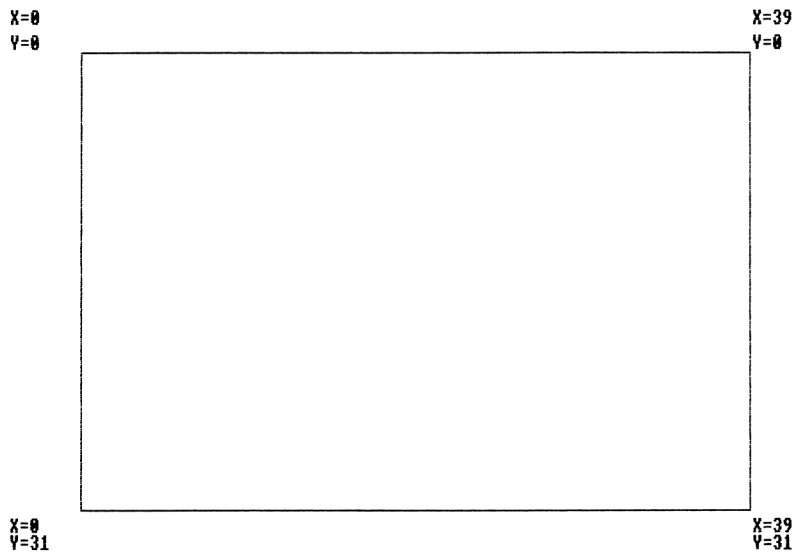


Fig. 1.3. Text coordinates, mode 1.

of rows down the screen from the top. Note that X=ϕ for the first position at the left edge of the screen and Y=ϕ for the top row of the screen.

In mode 1 the top left corner is ϕ,ϕ, the top right is ϕ,39, bottom left is ϕ,31 and bottom right 39,31. This is shown in Fig. 1.3.

On the BBC Micro the range of values that can be used for X and Y will depend upon the display mode selected. Modes 0 and 3 allow X values from ϕ to 79 and modes 1, 4, 6 or 7 will allow X from ϕ to 39. In modes 2 and 5 the highest value for X is 19. The value of Y can be from ϕ to 24 in modes 3, 6 or 7 and from ϕ to 31 for all other modes.

Graphics cursor

When used for graphics displays the BBC Micro uses a graphics cursor, but unlike the text cursor this is not displayed on the screen. The computer does, however, keep a record in memory of the values of X and Y coordinates for the graphics cursor and when lines are drawn they run from the current position of the graphics cursor to a new X,Y value specified in the drawing command.

Whereas the text coordinates have different ranges depending upon the display mode selected, the graphics cursor uses the same range of coordinates for all modes. The value of X indicates the position along a horizontal line starting at the left edge of the screen

and its range is from Ø to 1279, giving a total of 128Ø possible positions. The Y coordinate in graphics is measured up from the bottom of the screen and runs from Ø to 1Ø23 to give 1Ø24 possible points in the vertical direction.

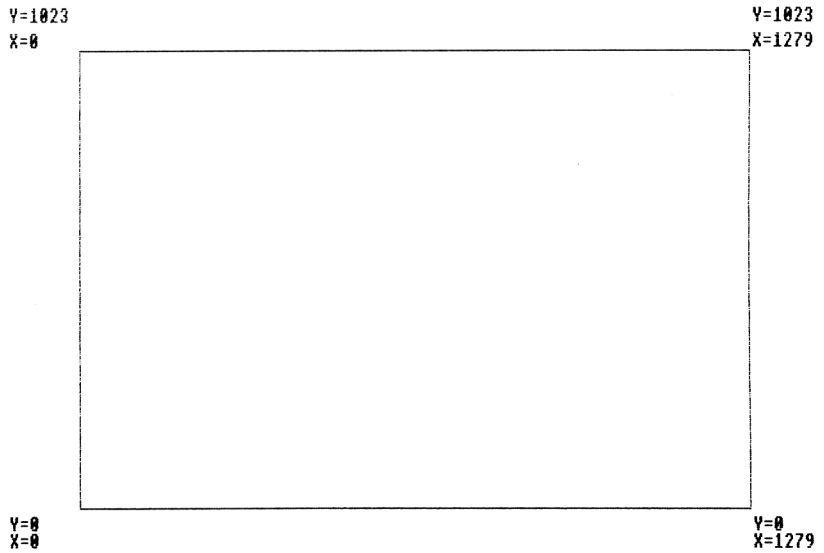


Fig. 1.4. Graphics coordinates.

Fig. 1.4 shows the layout of the screen coordinates for the high resolution graphics modes on the BBC Micro.

Moving the graphics cursor

As with the text cursor we can move the graphics cursor around on the screen and the command for doing this is MOVE, which takes the form

```
100 MOVE X, Y
```

where X and Y are the graphics coordinates of the point to which we want to move the cursor. At the start of operations after the mode has been selected the graphics cursor is always located at position Ø,Ø at the bottom left-hand corner of the screen.

Drawing lines

To produce a graphics picture on the screen we need to be able to

draw lines and this is achieved simply by using the DRAW command. Like the move command this has coordinates X and Y, which specify the position to which the line will be drawn. After the line has been drawn on the screen the graphics cursor will move to the new value of X,Y specified in the DRAW command. If the line is drawn at an angle to the horizontal or vertical axes of the screen the computer will automatically compute which dots on the screen have to be lit to produce the line. When the lower resolution modes are selected it will be noticed that a diagonal line is made up from a number of short horizontal or vertical lines. As the resolution is increased this stepped appearance of the lines becomes less obvious.

Let us now see if we can draw lines on the screen. For this we use the command DRAW and the form of the instruction will be

```
100 DRAW X,Y
```

Now a line will be drawn on the screen from the cursor position to the new position X,Y. If this instruction is the first graphics instruction in the program then the line will start from the origin point Ø,Ø.

Let us try this out with the following little program:

```
10 MODE 2
20 DRAW 100,100
```

This will produce a short line running from the bottom left corner towards the middle of the screen. If we now add the line

```
30 DRAW 100,700
```

and re-run the program this will produce a vertical line running upwards from the end of the first line. By adding the instruction

```
40 DRAW 700,700
```

a horizontal line is drawn to the right from the top of the vertical line.

Now the DRAW command always draws the new line starting from the last set of coordinates X and Y specified in the program, and if the last instruction was DRAW this means the line runs from the end of the last line to be drawn.

Sometimes we may want to draw a separate line at some other point on the screen. To do this we would use a MOVE instruction to move the cursor to the point where we want to start drawing the separate line.

Let us add some more instructions to our small program as follows, and then re-run it.

```
50 MOVE 800,100
60 DRAW 1000,100
```

This now produces a short horizontal line in the lower right-hand side of the screen.

A simple sketching program

By moving the cursor and drawing lines we can build up a picture on the screen, but this can become quite a tedious exercise. We need to work out the coordinates of all the points between which we are going to draw lines and then we have to write a program with the appropriate sequence of MOVE and DRAW commands to make the computer trace out the picture.

```
100 REM Simple Sketching Program
110 REM for BBC model B with OS1.0
120 MODE1
130 *FX4,1
140 VDU23,1,0;0;0;0;
150 X=600
160 Y=500
170 PLOT69,X,Y
180 PRINT TAB(0,0) "X= ";X;"  Y= ";Y
190 FOR N=1 TO 10000
200 PLOT71,X,Y
210 A = GET
220 IF A=137 THEN X=X+5
230 IF A=136 THEN X=X-5
240 IF A=139 THEN Y=Y+5
250 IF A=138 THEN Y=Y-5
260 IF A=85 THEN P=0
270 IF A=68 THEN P=1
280 IF A=81 THEN 350
290 IF P=0 THEN MOVE X,Y
300 IF P=1 THEN DRAW X,Y
310 PLOT69,X,Y
320 PRINT TAB(0,0) "X= ";X;"  Y= ";Y
330 FOR T=1 TO 100:NEXT T
340 NEXT N
350 VDU23,1,1;0;0;0;
360 *FX4,0
370 END
```

Fig 1.5. Sketching program.

Another approach to drawing pictures is to turn the computer into a sort of sketch pad by using the program listed in Fig. 1.5. To

control the movement of the cursor we can use the arrow keys and then we might use the keys U and D to indicate whether the pen is up or down. If the pen is up the cursor just moves to a new point but if the pen is down a line is drawn as the pen moves. When the pen is up nothing is drawn but in order to see where the cursor is, a PLOT command is used which draws a dot at the cursor point. This dot is erased before each new move. We shall be looking more closely at this PLOT command in the next chapter.

Now a few notes about the program itself. First, the program is written for a Model B machine or a Model A expanded to 32K memory. If you have a standard Model A the mode selected in line 12Ø should be MODE 4.

Line 13Ø disables the normal function of the arrow keys and allows them to produce an ASCII number for the GET command. Arrow keys are restored to their normal function in line 36Ø at the end of the program.

Line 14Ø switches off the text cursor and it is restored at the end of the program by line 35Ø. Note that this only works with operating system OS1.Ø or above. If you have the older operating system (OSØ.1) then line 14Ø must be changed to

```
140 VDU23,8202;0;0;0
```

and line 35Ø must be changed to

```
350 MODE1
```

or

```
35Ø MODE4
```

for the Model A.

Lines 18Ø and 32Ø print out the current X and Y coordinates at the top of the screen. Lines 19Ø–34Ø form the main drawing loop.

Lines 22Ø–25Ø check for the four arrow keys and increase or decrease the values of X and Y according to which key is pressed. Lines 26Ø and 27Ø check for keys U and D, and P is used to indicate the state of the pen. Line 28Ø checks for the Q key which causes the program to end.

Lines 29Ø and 3ØØ execute MOVE or DRAW according to the state of P and line 22Ø puts a dot at position X,Y. Line 31Ø puts a dot at the X,Y position, whilst line 2ØØ erases the dot before moving to the next cursor position.

Line 33Ø is merely a time delay, the size of which can be adjusted to control the speed of movement of the pen.

Chapter Two
Drawing and Plotting

In the last chapter we developed a simple program which enabled us to use the computer display as a form of sketch pad, and it is quite possible to produce simple drawings with this program. However, you will have found that drawing shapes such as polygons or circles is not at all easy and even simple shapes such as squares and rectangles need some care to produce good results.

If we are interested in producing technical drawings or if we need to draw a number of shapes rapidly then it would be useful to have routines by which the computer draws the shape itself having been given sufficient information to define exactly what it has to draw. With such a routine it might be possible to draw circles of any desired size at any specified point on the screen. In fact we can effectively arrange to have the computer act as a special kind of paint brush, or perhaps more accurately a rubber stamp, which will produce any desired shape as many times as we want it and where we want it.

Let us now take a look at the techniques required for drawing various common shapes such as squares, rectangles, polygons, circles and ellipses. For a start we'll take a look at the simple square figure.

Drawing squares

One simple solution to drawing a square is to work out the X,Y coordinates for each of the four corners of the square and then use a MOVE command to place the cursor at say the bottom left corner of the square. At this point four DRAW commands may be used to draw in the four lines making up the sides of the square.

Suppose we want to draw a square with sides 1ØØ units long and have it positioned with its lower left-hand corner at position X,Y =

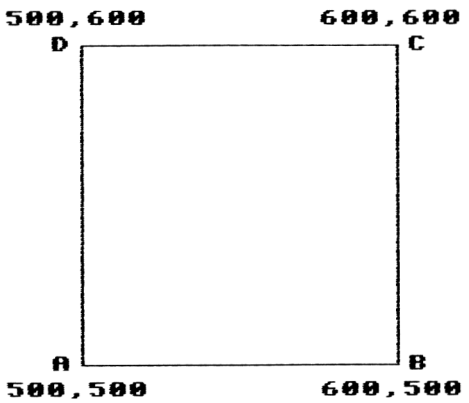


Fig. 2.1. Actual coordinates for a square.

5ØØ,5ØØ. This is shown in fig. 2.1. The coordinates of the first corner (A) at the bottom left are 5ØØ,5ØØ. The other lower corner (B) is 1ØØ units further to the right, so the X value increases by 1ØØ to 6ØØ but the Y value remains at 5ØØ. The two upper corners (C and D) have the same pair of values for X but their Y values are increased by 1ØØ, giving points 6ØØ,6ØØ and 5ØØ,6ØØ for C and D respectively. Now to draw the square we could use the following little program.

```
100 REM DRAW SQUARE
110 MODE 1
120 MOVE 500,500
130 DRAW 600,500
140 DRAW 600,600
150 DRAW 500,600
160 DRAW 500,500
170 END
```

Drawing more squares

If we want to draw lots of squares of different sizes at various points on the screen the process of working out all of the coordinates for each square becomes rather tedious. A much better approach is to use a square drawing procedure or subroutine which will work out the corner points and draw the squares automatically.

To work out the required coordinates we need the location of a reference point on the square such as the bottom left-hand corner and the width of the required square. Suppose the reference point is X,Y and the width is W, then the required square is as shown in Fig. 2.2. Point A of the square is simply X,Y. For point B the width W is

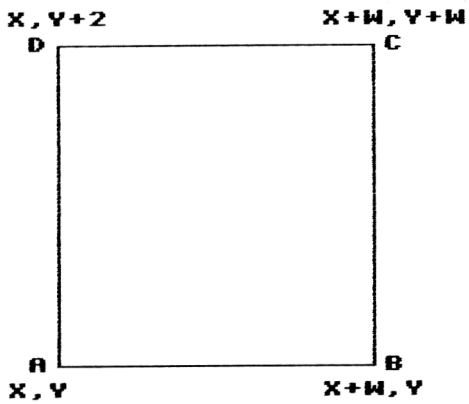


Fig. 2.2. Coordinates for drawing a square using X, Y and W.

added to X to give coordinates X+W,Y. Points C and D both have width W added to Y, giving C = X+W,Y+W and D = X, Y+W.

All we need do now is set up the bottom left corner coordinates and the width of the square in the main program. In the procedure the width is added to X and Y as required to calculate the other three corners and lines are then drawn for the sides of the square.

```
100 REM Random squares
110 MODE 1
120 FOR N=1to30
130 X=RND(1200)
140 Y=RND(1000)
150 W= 40+200*RND(1)
160 PROCSQUARE
170 NEXT N
180 END

500 DEFPROCSQUARE
510 MOVE X,Y
520 DRAW X+W,Y
530 DRAW X+W,Y+W
540 DRAW X,Y+W
550 DRAW X,Y
560 ENDPROC
```

Fig. 2.3. Program to draw random squares.

Fig. 2.3 shows a program in which a set of 30 squares of different sizes are drawn at random points on the screen. The pattern drawn should be different each time this program is run. Note that here a procedure has been used for the actual process of drawing each square. This could of course equally well have been achieved by using a subroutine for drawing the square. Fig. 2.4 shows a typical display from this program.

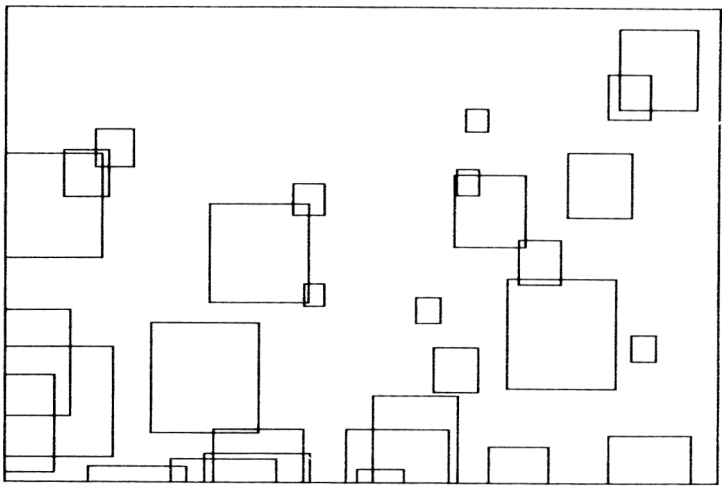


Fig. 2.4. Display produced by random squares program.

Drawing rectangles

A more general form of a four-sided figure is the rectangle, where the width and height are different. We can quite easily adapt our square drawing routine to make it draw rectangles instead. In fact this new routine could equally well be used for drawing squares by simply making the width and height values the same.

In the new procedure H is added to the Y coordinate whilst W is now added only to the X coordinate. A program for drawing random rectangles all over the screen would be as shown in Fig. 2.5.

```
100 REM Random rectangles
110 MODE 1
120 FOR N=1 TO 30
130 X=RND(1200)
140 Y=RND(1000)
150 W=10+90*RND(1)
160 H=10+90*RND(1)
170 PROCRECT
180 NEXT N
190 END

500 DEFPROCRECT
510 MOVE X,Y
520 DRAW X+W,Y
530 DRAW X+W,Y+H
540 DRAW X,Y+H
550 DRAW X,Y
560 ENDPROC
```

Fig. 2.5. Random rectangles program.

Drawing hexagons

Drawing squares and rectangles is easy but suppose we want to draw a hexagon, which has six equal sides. The basic hexagon shape is

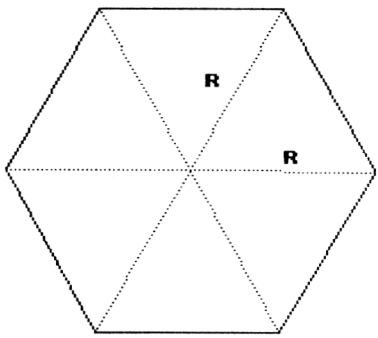


Fig. 2.6. Basic hexagon shape made up of six equal-sided triangles.

shown in Fig. 2.6. A point to note is that if we draw lines between opposite points on the hexagon they all pass through the centre and all have the same length. In effect the hexagon is made up from six equal sided triangles. To draw the hexagon we need to get involved in some simple trigonometry.

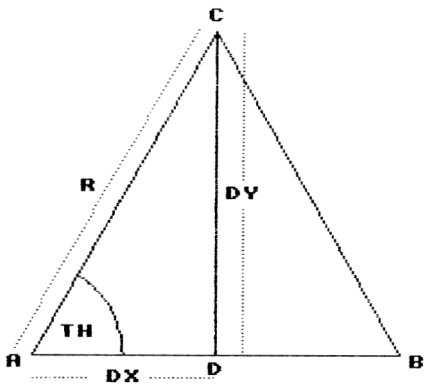


Fig. 2.7. One of the triangles forming the hexagon.

In Fig. 2.7 one of the triangles forming the hexagon has been drawn with one of its sides along the X axis. Suppose we choose the centre of our hexagon (point A) as the reference point, then the coordinates of point A will be X,Y. Now let us make the width (W) value equal to the length of side AB. Since sides AB and AC are equal then side AC is also of length W. The first point for drawing will be point B and its coordinates will be X+W,Y.

To draw the side BC we need to know the coordinates of point C.

Let us drop a vertical line from C to point D. To find and X value for point C we need to know the length AD and to find the Y value we need to know the length CD. This is where the trigonometry comes in.

We will call the angle at point A of the triangle *theta*. The function we need to use now is sin(theta). The definition of sin(theta) is that it is the ratio of the length of the side of the triangle opposite angle theta to the length of the hypotenuse (the side opposite the right angle). So in our triangle

```
SIN(THETA) = CD/AC
```

We already know that AC = radius R and CD is the change we need to make to Y, which we will call DY. Therefore

```
SIN(THETA) = DY/R
```

and if we multiply both sides by R we get

```
DY = R * SIN(THETA)
```

Having found DY we need to find a value for AD which we shall call DX. Now it just happens that cos(theta) is the ratio of the length of the adjacent side (AD) to the length of the hypotenuse (AC), so we get

```
COS(THETA) = AD/AC = DX/R
```

and by rearranging the equation

```
DX = R * COS(THETA)
```

Another useful fact is that if theta is 0 then cos(theta) = 1 and sin(theta) = 0, so for our first point B the values of DX and DY are R and 0 respectively. So for both points B and C we simply have to calculate values for DX and DY, then add them to the X and Y values for the centre point. The same rule applies to all of the points of the hexagon.

How do we decide on a value for theta? Well there are 360 degrees in a complete rotation of the angle theta, and since we have six sides each triangle must add 360/6 or 60 degrees to the value of theta. Our computer, however, doesn't work in degrees but uses radians instead. All we need to know here is that there are 2 × pi radians in 360 degrees, so therefore the angle increases by 2 × pi/6 radians for each point of the hexagon. To draw our hexagon we simply use a loop to repeat the calculations and draw a side six times. The program in Fig. 2.8 draws lots of varying sized hexagons all over the screen.

```
100 REM Random hexagons
110 MODE1
120 FOR NH = 1 TO 50
130 X= RND(1200):Y=RND(1000)
140 R= 30+RND(150)
150 PROCHEX
160 NEXT NH
170 END

500 DEFPROCHEX
510 MOVE X+R,Y
520 DTH = 2 * PI/6
530 TH=0
540 FOR N=1 TO 6
550 TH=TH+DTH
560 DX = R*COS(TH)
570 DY = R*SIN(TH)
580 DRAW X+DX,Y+DY
590 NEXT N
600 ENDPROC
```

Fig. 2.8. Random hexagons program.

Here DTH is the change required in the value of TH for each step and we have shortened the name THETA to TH. Note the MOVE to get the cursor to the first point to be drawn on the hexagon rather than to the centre of the hexagon.

Drawing polygons

A hexagon is an example of a many-sided figure for which the general name is *polygon*. Now suppose we want to draw a polygon with a different number of sides.

In the case of the six-sided hexagon the angle for each point increased by 2 × pi/6. If we wanted to have eight sides then the total angle of 2 × pi radians would have to be divided up into eight equal parts, so the angle theta increases by 2 × pi/8 radians for each point. To make a general polygon drawing routine we could add a new parameter NS (number of sides) and then modify the procedure to make the required number of drawing steps. To see how this works try running the program shown in Fig. 2.9.

This program will draw a series of figures of increasing size, centred on a point near the centre of the screen, as shown in Fig. 2.10. The figures have an increasing number of sides, starting with a triangle and working up to a twelve-sided figure. We could modify this to produce a random polygon drawing program as shown in Fig. 2.11.

```
100 REM Polygon drawing
110 MODE1
120 X = 600:Y = 500
130 FOR NS = 3 to 12
140 R = 40*NS
150 PROCPOLY
160 NEXT NS
170 END

500 DEFPROCPOLY
510 MOVE X+R,Y
520 DTH = 2*PI/NS
530 TH=0
540 FOR N = 1 TO NS
550 TH=TH+DTH
560 DX = R*COS(TH)
570 DY = R*SIN(TH)
580 DRAW X+DX,Y+DY
590 NEXT N
600 ENDPROC
```

Fig. 2.9. Polygon drawing program.

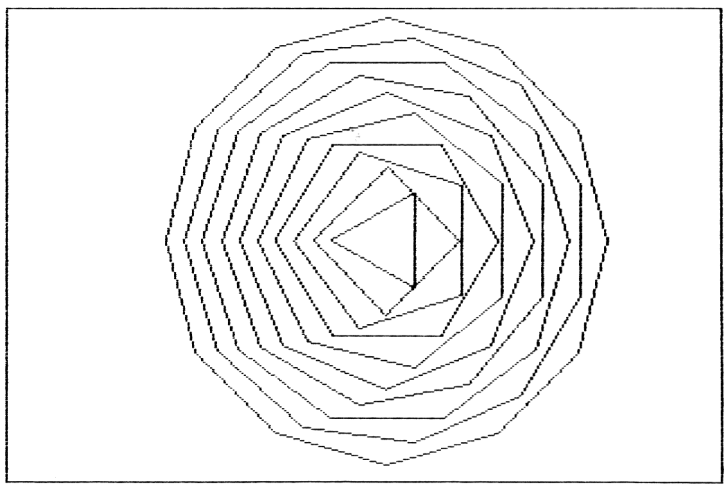


Fig. 2.10. Display produced by program in Fig. 2.9.

Here the minimum number of sides has been set at three since this gives a triangle. A two-sided figure would just be a straight line. The results on the screen will be similar to Fig. 2.12.

```
100 REM Random polygons
110 MODE 1
120 FOR NP=1 TO 30
130 X=RND(1200):Y=RND(1000)
140 R=30+RND(150)
150 NS=2+RND(9)
160 PROCPOLY
170 NEXT NP
180 END

500 DEFPROCPOLY
510 MOVE X+R,Y
520 DTH=2*PI/NS
530 TH=0
540 FOR N=1 TO NS
550 TH=TH+DTH
560 DX=R*COS(TH)
570 DY=R*SIN(TH)
580 DRAW X+DX,Y+DY
590 NEXT N
600 ENDPROC
```

Fig. 2.11. Program to draw a set of random size polygons.

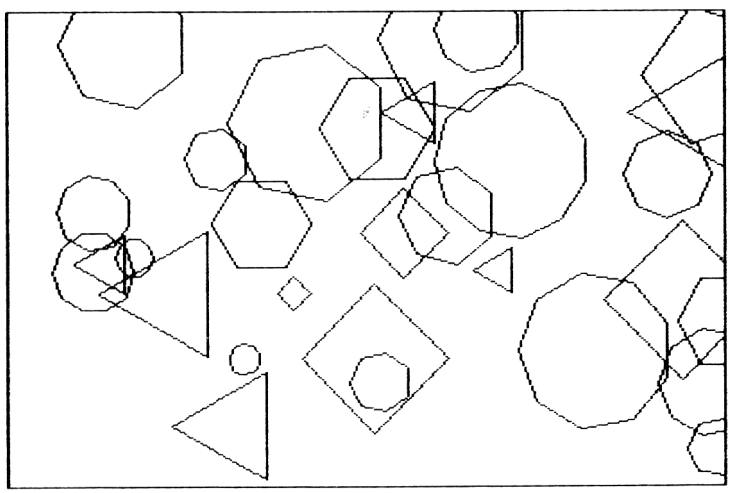


Fig. 2.12. Display produced by random polygon program.

Drawing circles

Rectangles and squares are fairly easy to draw, but one figure that we shall regularly want to produce is the circle and here things are much more complex. There are in fact three basic approaches to the

drawing of circles using a computer and we shall take a look at each of them.

Producing the curved line that makes up a circle is actually achieved by drawing a large number of very short straight lines at slightly different angles and these link up to give the appearance of a circle on the screen. The more short lines or dots there are the better the circle will look. Some circle drawing techniques operate faster than others but the slower techniques have advantages when we want to divide up a circle.

If you use the polygon drawing routine and draw more and more sides to the polygon you will notice that it becomes more and more like a circle, so obviously one method for producing circles might be based on the polygon drawing routine.

To draw our circle we merely calculate a series of values of X1 and Y1 for values of theta from 0 to 360 degrees (0 to 2 × pi radians). The number of points we need depends upon the size of the circle and how accurately we want to draw it. A good figure to use is the number of units of the radius, so for a circle of radius 50 we might use 50 points. The angle theta for each step can be found by simply dividing 2 × pi by the required number of points, so the step size would be pi × 2/R. A procedure for drawing a circle using this technique would be as follows.

```
500 DEFPROCCIRCL
510 MOVE X+R,Y
520 DT=PI*2/R
530 TH=0
540 FOR N=1 TO R
550 TH=TH+DT
560 DRAW X+R*COS(TH),Y+R*SIN(TH)
570 NEXT N
580 ENDPROC
```

Because of the SIN and COS functions this circle drawing method can be fairly slow, especially for large size circles where a lot of points have to be calculated. It does, however, have the advantage that it allows us to draw segments of a circle (arcs) quite easily compared with the other circle drawing methods.

Drawing arcs

To draw a quarter-circle we can use a modified version of the circle drawing technique just described. For the upper right quarter we

could set up a FOR NEXT loop with a variable TH to run from TH = ∅ to TH = PI/2 with a step size of 2*PI/R. For a half-circle the loop could run from ∅ to PI. This would in fact give the upper half of the circle. To get the lower half we set the loop to run from TH = PI to TH = 2*PI. Note that here we must change the MOVE instruction which moves the cursor to the first point that we want to draw so it would now become

```
510 MOVE X+R*COS(PI),Y+R*SIN(PI)
```

Using a similar technique we could draw any of the four quarters of the circle or for that matter any segment of the circle that we like. Simply choose the start and finish angles and move the cursor to the start angle position before going into the drawing loop. Two extra variables are now required (AS and AF) to define the start and finish angles for the arc. This gives a general purpose arc or circle drawing routine as follows.

```
500 DEFPROCARC
510 MOVE X+R*COS(AS),Y+R*SIN(AS)
520 DT=2*PI/R
530 FOR TH=AS TO AF STEP DT
540 DRAW X+R*COS(TH),Y+R*SIN(TH)
550 NEXT TH
560 ENDPROC
```

where AS and AF are the start and finish angles for the arc measured in radians from a zero along the X axis to the right of the origin point CX,CY as shown in Fig. 2.13. Note that the values for AS and AF must be set up in the main program before calling the procedure to draw the arc.

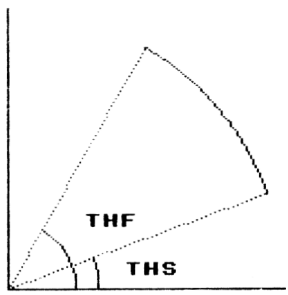


Fig. 2.13. Start and finish angles for drawing an arc.

If you want to work in degrees, and most of us are more familiar with degrees, then these must be converted to radians by using

```
Angle(rads) = Angle(degs)*PI/180
```

The quadratic equation method

Calculating all those sin and cos functions makes the drawing of circles quite slow, because sin and cos functions are relatively slow in execution. An alternative technique for drawing circles makes use of a different mathematical formula for a circle.

If we consider the triangle in Fig. 2.7 with sides of R, DX and DY there is another way in which the three sides are related. For a right-angled triangle the square of the length of the longest side is the sum of the squares of the lengths of the other two sides. This is our friend Pythagoras's famous theorem.

So now we get for our triangle

```
(AC)*(AC) = (AD)*(AD) + (CD)*(CD)
```

or alternatively putting in the variables we used for those sides we get

```
R * R =  (DX * DX) + (DY * DY)
```

and this can be re-arranged to give us

```
DY * DY = (R * R) - (DX * DX)
```

from which we can get DY by simply taking the square root.

So our calculation for DY now becomes

```
DY = SQR((R * R) - (DX * DX))
```

Here we have an equation that doesn't involve sines, cosines or angles. Now SQR is also a relatively slow operation but we only have to do it once per step so the calculation and drawing should be a little quicker than for the method using SIN and COS.

The values of DX and DY in this equation are measured with reference to the centre point of the circle. To place the circle at some particular point on the screen we shall have to add in the X and Y coordinates for the point where the circle is to be drawn. To avoid confusion we shall call these coordinates CX and CY.

To calculate all the points around the circle we need to calculate values of DY for a series of values of DX ranging from $-R$ to $+R$ and the more points we calculate the better the circle will look. A convenient arrangement is to make the number of points equal to the value of R.

Let us say the circle is going to have a radius R of 50 units and be

placed at point X,Y = 500,500, then the program required to draw it will be

```
100 REM Circle quadratic method
110 MODE 1
120 X=500:Y=500:R=50
130 MOVE X-R,Y
140 FOR DX=-R TO R
150 DY=SQR(R*R-DX*DX)
160 DRAW X+DX,Y+DY
170 NEXT DX
180 MOVE X-R,Y
190 FOR DX=-R TO R
200 DY=SQR(R*R-DX*DX)
210 DRAW X+DX,Y-DY
220 NEXT DX
230 END
```

The drawing is done in two steps, since the first loop will draw only the upper half-circle so a second loop plotting Y−DY is required. We can of course make this into a procedure or subroutine in the same way as for squares and rectangles and produce a program that will draw various size circles all over the screen. Fig. 2.14 is a listing of the program and Fig. 2.15 shows the results obtained when it is run.

```
100 REM Random circles (quadratic)
110 MODE 1
120 FOR N=1 TO 30
130 X=RND(1200)
140 Y=RND(1000)
150 R=10+90*RND(1)
160 PROCCIRCL
170 NEXT N
180 END

500 DEFPROCCIRCL
510 MOVE X-R,Y
520 FOR DX=-R TO R
530 DY=SQR(R*R-DX*DX)
540 DRAW X+DX,Y+DY
550 NEXT DX
560 DRAW X+R,Y
570 MOVE X-R,Y
580 FOR DX=-R TO R
590 DY=SQR(R*R-DX*DX)
600 DRAW X+DX,Y-DY
610 NEXT DX
620 DRAW X+R,Y
630 ENDPROC
```

Fig. 2.14. Program to draw random size circles.

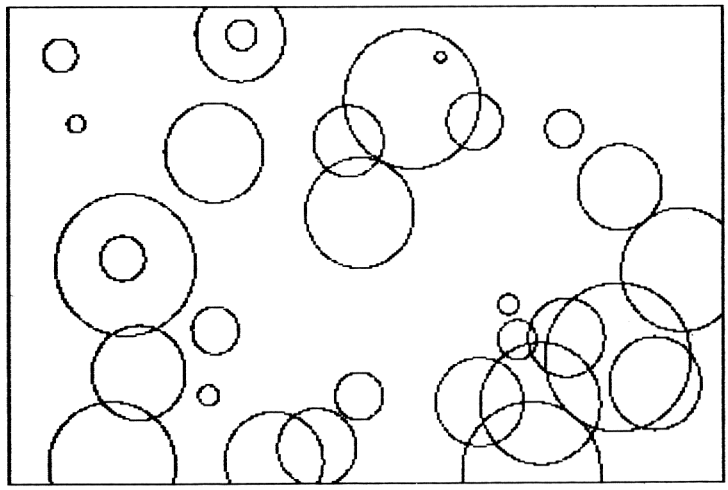


Fig. 2.15. Display of random size circles produced by program of Fig. 2.14.

With this routine the number of calculations depends upon the size of the circle, and it will be seen that the larger circles take a noticeable time to draw. This is because the computer has quite a lot of calculations to carry out. The square root function itself is rather slow in BASIC. Things could be speeded up slightly by calculating R*R outside the drawing loop and using the result in the calculation for DY. This saves some multiplication operations but the overall calculation is still quite slow. If we want to draw circles faster we will need to look at other ways of calculating the points around the circle.

The rotation method

A different approach to the calculation of the X,Y values for a circle is to base them upon the angle through which the radial line is rotated at each step. In this case the new values for X and Y are calculated from the previous values rather than from the radius and the total angle.

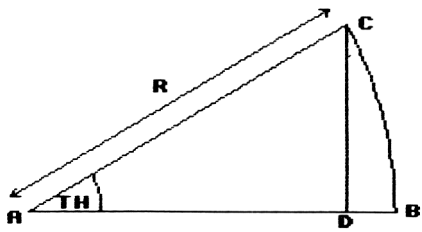


Fig. 2.16. Rotation of a point from X axis.

If we look at Fig. 2.16 the initial value of Y is zero, so that only the original X value, which also happens to be equal to R, affects the results. Here we get

```
X1 = X * COS(TH)
Y1 = X * SIN(TH)
```

Now consider the situation where the radial line is vertical and is moved through angle theta. This is shown in Fig. 2.17. Here the initial value of X is 0 and only the Y value affects the results. In this case the value of X1 is negative since the X point has been shifted to the left of the line where X=0. Here we get the results

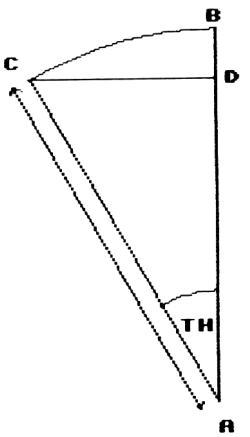


Fig. 2.17. Rotation of a point from the Y axis.

```
X1 = -Y * SIN(TH)
Y1 =  Y * COS(TH)
```

If we combine these two results we can produce a general expression for calculating X1 and Y1 for any initial values of X and Y. The two new equations are

```
X1 = X * COS(TH) - Y * SIN(TH)

Y1 = X * SIN(TH) + Y * COS(TH)
```

Now the big advantage of this approach is that the value of TH is constant so we can work out the values of SIN(TH) and COS(TH) before entering our coordinate calculation loop, thus eliminating virtually all of the trigonometric calculations which tend to be slow. The procedure for drawing a circle now becomes:

```
500 DEFPROCCIRCL
510 MOVE X+R,Y
520 TH=PI*2/R
530 CT=COS(TH)
```

```
540 ST=SIN(TH)
550 DX=R:DY=0
560 FOR N=1 TO R
570 X1=DX*CT-DY*ST
580 DY=DX*ST+DY*CT
590 DX=X1
600 DRAW X+DX,Y+DY
610 NEXT N
620 ENDPROC
```

This method can also be used for drawing arcs, but is a little more difficult to set up so the trigonometric method is usually better suited for that purpose.

Drawing polygons by rotation

Now if we can draw circles using this line rotation technique then it should be possible to draw hexagons and polygons as well. For a hexagon the angle theta will be $2 \times \text{pi}/6$ so a procedure for drawing hexagons becomes:

```
500 DEFPROCHEX
510 DX=R:DY=0
520 MOVE X+R,Y
530 ST=SIN(PI/3):CT=COS(PI/3)
540 FOR N=1 TO 6
550 X1=DX*CT-DY*ST
560 DY=DX*ST+DY*CT
570 DX=X1
580 DRAW X+DX,Y+DY
590 NEXT N
600 ENDPROC
```

As before we can develop this little routine into a general polygon drawing routine by adding the variable NS (number of sides) and we can produce a program which draws polygons with from 3 to 12 sides in various sizes all over the screen. This is shown in Fig. 2.18.

This procedure for drawing polygons can be used as a general method in any program. Note that it will draw squares and triangles but the triangles will always be equal-sided ones.

Star shaped figures and wheels

The polygon drawing routine can easily be modified to draw star shaped figures. In this case a line is drawn from the centre to each

```
100 REM Random polygons by rotation
110 MODE1
120 FOR NP=1 TO 30
130 X=RND(1200):Y=RND(1000)
140 R=25+RND(100):NS=3+RND(9)
150 PROCPOLY
160 NEXT N
170 END

500 DEFPROCPOLY
510 TH=2*PI/NS
520 CT=COS(TH):ST=SIN(TH)
530 DX=R:DY=0
540 MOVE X+DX,Y+DY
550 FOR N=1 TO NS
560 X1=DX*CT-DY*ST
570 DY=DX*ST+DY*CT
580 DX=X1
590 DRAW X+DX,Y+DY
610 NEXT N
620 ENDPROC
```

Fig. 2.18. Program for producing polygons by rotation.

calculated point of the polygon by changing the drawing procedure. The program listing in Fig. 2.19 will draw a pattern of random star shaped figures on the screen.

```
100 REM Random stars by rotation
110 MODE1
120 FOR NP=1 TO 30
130 X=RND(1200):Y=RND(1000)
140 R=25+RND(100):NS=3+RND(9)
150 PROCSTAR
160 NEXT N
170 END

500 DEFPROCSTAR
510 TH=2*PI/NS
520 CT=COS(TH):ST=SIN(TH)
530 DX=R:DY=0
540 FOR N=1 TO NS
550 MOVE X,Y
560 DRAW X+DX,Y+DY
570 X1=DX*CT-DY*ST
580 DY=DX*ST+DY*CT
590 DX=X1
600 NEXT N
610 ENDPROC
```

Fig. 2.19. Program for producing random stars.

In this case the graphics cursor is moved back to the centre before each new point of the figure is calculated so the line radiates from the centre like the spoke of a wheel.

Wheel shapes can be drawn by first drawing the star and then drawing a circle of the same radius around the same centre point.

The PLOT command

So far we have been drawing using the MOVE and DRAW commands which are simple to use, but there is another much more powerful command available for drawing on the BBC Micro. This is the PLOT command. It has many variations which allow moving the cursor, drawing lines, both solid and dotted, plotting individual points and even filling in areas of the screen.

The general format of the PLOT command is

```
100  PLOT n, x, y
```

where N is a number which determines the type of operation the PLOT command will actually carry out and X,Y are the coordinates. An important difference with the PLOT command, however, is the way in which the coordinate values X and Y are actually interpreted.

The command PLOT 4, X,Y moves the cursor to the point X,Y and is directly equivalent to the MOVE command that we have been using so far. Similarly, PLOT 5,X,Y draws a line in exactly the same way as the DRAW command.

PLOT Ø,X,Y also moves the cursor around the screen, but the important difference between it and the MOVE or PLOT4 command is that the coordinates X,Y in this case are measured relative to the current position of the cursor. Thus if the cursor is at a position 5ØØ,5ØØ on the screen and we use

```
PLOT 0,100,100
```

the cursor will actually move to a point 6ØØ,6ØØ and the effect is the same as using MOVE 6ØØ,6ØØ. In the same way PLOT 1,X,Y causes a line to be drawn from the current cursor position to a point X,Y measured relative to the current position.

What advantage is there in having this relative position action? Well, if you remember when we drew squares and rectangles we had to add the X,Y values of the corner to the calculated values for the sides in order to place the rectangle where we wanted it on the screen.

By using the relative plotting statements we can make the rectangle drawing procedure much simpler, as follows:

```
500 DEFPROCRECT
510 MOVE X,Y
520 PLOT1,W,0
530 PLOT1,0,H
540 PLOT1,-W,0
550 PLOT1,0,-H
560 ENDPROC
```

At present the rectangles are drawn with the X,Y point located at the bottom left-hand corner, but to be more consistent with the way we have drawn our circles and polygons it might be desirable to have the X,Y reference point at the centre of the rectangle. This can readily be achieved by using a slightly different sequence of drawing instructions where the move statement has an offset of half width and half height:

```
500 DEFPROCRECT
510 MOVE X-W/2,Y-H/2
520 PLOT1,W,0
530 PLOT1,0,H
540 PLOT1,-W,0
550 PLOT1,0,-H
560 ENDPROC
```

Drawing diamonds

If we modify the rectangle drawing procedure slightly we can draw diamond shapes. In this case we can move the cursor by half the width to pick up the right-hand corner of the diamond and then the plotting statements will each have half width and half height terms to draw lines to the other points of the diamond. The resultant procedure is

```
500 DEFPROCDIAM
510 MOVE X,Y
520 PLOT1,-W/2,H/2
530 PLOT1,-W/2,-H/2
540 PLOT1,W/2,-H/2
550 PLOT1,W/2,H,2
560 ENDPROC
```

Relative plotting for polygons and circles

We can use the relative plotting technique for circles and polygons. In this case we have to calculate the new values for DX and DY, which we might call X1 and Y1, and then we use the difference in the PLOT command. Finally, of course, the DX and DY values need to be updated.

For the trigonometric method the procedure would become

```
500 DEFPROCPOLY
510 DX=R:DY=0
520 PLOT0,DX,DY
530 DTH=2*PI/NS:TH=0
540 FOR N=1 TO NS
550 TH=TH+DTH
560 X1=R*COS(TH)
570 Y1=R*SIN(TH)
580 PLOT1,X1-DX,Y1-DY
590 DX=X1:DY=Y1
600 NEXT N
610 ENDPROC
```

For the rotation technique we would have

```
500 DEFPROCPOLY
510 TH=2*PI/NS
520 CT=COS(TH):ST=SIN(TH)
530 DX=R:DY=0
540 PLOT0,DX,DY
550 FOR N=1 TO NS
560 X1=DX*CT-DY*ST:Y1=DX*ST+DY*CT
570 PLOT1 X1-DX,Y1-DY
580 DX=X1:DY=Y1
590 NEXT N
600 ENDPROC
```

Note that we no longer need to use X and Y in the procedure provided that we remember to move the cursor to the desired centre point (X,Y) before going into the procedure.

Scaling and stretching

In drawing squares, polygons and circles the size of the displayed figure depends upon the value of W or R that we used in the drawing routine. Thus, by altering W or R we can alter the size or scale of the figure.

In the case of the rectangle there are two scaling figures, one for

width (W) and one for height (H). In effect we have a square which has been stretched or compressed in one direction. Assuming that we apply stretching only horizontally or vertically this just means that the X and Y scale values are different.

We could apply the stretching idea to other figures by putting in two extra variables which would be the X and Y scale factors. To achieve the correct results the reference point around which the figure is drawn should be at the centre of the figure. For polygons and circles this is always true in the drawing methods we have used. In this case the scale factors are used as multipliers for the DX and DY terms in the drawing calculations.

Let us consider a circle and we will use the trigonometric drawing method. Two new terms SX and SY are now used in the procedure, which becomes:

```
500 DEFPROCCIRCL
510 MOVE X+R*SX,Y
520 DTH=PI*2/R:TH=0
530 FOR N=1 TO R
550 TH=TH+DTH
560 DX=R*COS(TH):DY=R*SIN(TH)
570 DRAW X+DX*SX,Y+DY*SY
580 NEXT N
590 ENDPROC
```

If SX and SY are both 1 then the figure drawn will be a circle. If SX>1 or SY<1 the figure becomes an ellipse with the longer axis horizontal. If SX<1 and SY>1 the ellipse will have its long axis vertical. If SX or SY is negative this will simply have the effect that the figure is drawn backwards. If we had a figure that was not symmetrical then the left side would be displayed at the right, or the top would move to the bottom giving a mirror image effect.

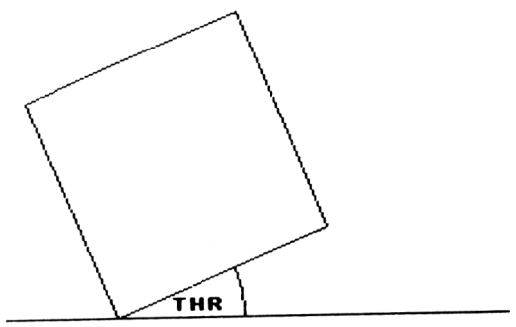


Fig. 2.20. A square rotated through an angle THR.

Rotation of figures

The figures we have produced so far have all been drawn with their X axis horizontal. Suppose, however, we want to draw a square but have it displayed tilted at an angle as shown in Fig. 2.20. This can be done by using the same basic technique as in the rotation method for drawing circles. In this case the rotation equations are applied to each point on the figure in turn to calculate its new position for the rotated figure.

To find the rotated values for DX and DY we use

```
X1 = DX*COS(THR) - DY*SIN(THR)
```

and

```
Y1 = DX*SIN(THR) + DY*COS(THR)
```

where DX and DY are the coordinates of each point measured relative to the point about which we want to rotate the figure. X1

```
100 REM Rotated rectangles
110 MODE1
120 X=640:Y=512:W=300:H=100
130 THR=0:DTH=PI/6
140 FOR N=1 TO 6
150 THR=THR+DTH
160 MOVE X,Y
170 DX=W:DY=0
180 PROCROT
190 DRAW X+DX,Y+DY
200 DX=W:DY=H
210 PROCROT
220 DRAW X+DX,Y+DY
230 DX=0:DY=H
240 PROCROT
250 DRAW X+DX,Y+DY
260 DX=0:DY=0
270 PROCROT
280 DRAW X+DX,Y+DY
290 NEXT N
300 END

700 DEFPROCROT
710 XR=DX*COS(THR)-DY*SIN(THR)
720 YR=DX*SIN(THR)+DY*COS(THR)
730 DX=XR:DY=YR
740 ENDPROC
```

Fig. 2.21. Program for producing rotated rectangles.

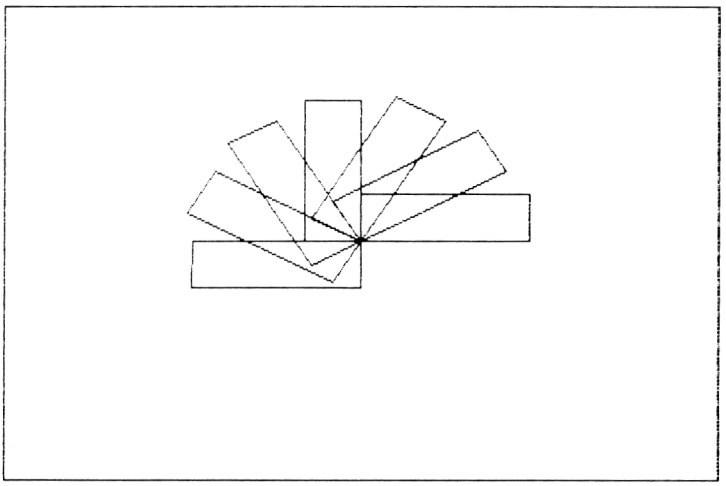


Fig. 2.22. Rotated rectangles.

```
100 REM Rotated squares using table
105 REM of X,Y values.
110 MODE1
120 DIM X(4),Y(4)
130 X(1)=300:Y(1)=0
140 X(2)=300:Y(2)=300
150 X(3)=0:Y(3)=300
160 X(4)=0:Y(4)=0
170 DTH=PI/6:TH=0:CX=600:CY=500
180 FOR NR=1 TO 6
190 MOVE CX,CY
200 THR=THR+DTH
210 FOR N=1 TO 4
220 DX=X(N):DY=Y(N)
230 PROCROT
240 DRAW CX+DX,CY+DY
250 NEXT N
260 NEXT NR
270 END

700 DEFPROCROT
710 XR=DX*COS(THR)-DY*SIN(THR)
720 YR=DX*SIN(THR)+DY*COS(THR)
730 DX=XR:DY=YR
740 ENDPROC
```

Fig. 2.23. Program for producing rotated squares.

and Y1 are the new values we use for drawing. The angle of rotation is THR radians relative to the positive X axis.

Let us start with a square and assume that we are using the bottom left-hand corner of the square as a reference point about which it will

be rotated. The program in Fig. 2.21 draws a series of rectangles rotated successively in 30 degree steps to give the display shown in Fig. 2.22.

This scheme could be extended to any figure by first having a table of values of DX and DY for the points in the figure and then setting DX and DY to each pair of values in turn before calling PROCROT and drawing the line. So for our square we first set up a table of values X(N) and Y(N) as shown in Fig. 2.23.

Here since we have used X and Y arrays to define the points in the figure the variables CX and CY have been used to defined the origin point around which the figure will be drawn on the screen.

Note that if stretching and rotation procedures are used they must be performed on the DX,DY values after they have been calculated and before the lines are drawn. The results will also be different according to the order in which the stretch and rotate operations are carried out, since the stretch operation is applied only along the X or Y axis of the figure.

Chapter Three

Painting by Numbers

So far our pictures and text on the screen have all been displayed in black and white, since these are the colours that the BBC Micro initially selects for all modes of operation. Now the real world is coloured and any artist would be rather limited in his activities if he had only black and white paints to work with. Assuming that a colour television receiver or colour TV monitor is available we can produce much more attractive displays by making use of the colour capability of the BBC Micro.

Even simple text displays can benefit from the use of colour. Different parts of the display can readily be picked out when they are presented in different colours. If a particular item needs to be brought to the attention of the viewer it can be emphasised by using a colour which contrasts with the colour used for the rest of the text. This is why bank statements and bills suddenly acquire red lettering when they are overdrawn or the payment is overdue.

When we come to games displays, of course, the enjoyment of the game is greatly enhanced when a coloured display is used. Colour will often add a sense of realism to a game display and it can be much easier to pick out targets and avoid the bombs when they are displayed in different colours.

Selecting a different text colour

As mentioned above, the BBC Micro normally selects white text or graphics on a black background at switch-on or after a mode change. We can change the colour of the text by using the command

```
COLOUR C
```

where C is a number between 0 and 15 which determines the new display colour. Let us try some experiments by typing in the following series of commands:

```
MODE2
COLOUR1
COLOUR2
COLOUR3
COLOUR4
COLOUR5
```

The first and second commands will be displayed in white as usual, but when the return key is pressed after COLOUR1 you will notice that the prompt arrow has changed colour to red and that as you type in new text it too is red. After the following commands the colours will change through green, yellow, blue and magenta. If you continue putting in higher numbers the numbers above 7 will produce flashing colours and if you go on with numbers above 16 the sequence of colours repeats. Note that if you type COLOUR0 you cannot see the following text. This is because 0 selects black which makes the text the same colour as the background and therefore invisible.

Now try the same thing again but use MODE1 to select the display mode. Now you will find that there are only four colours, including black, and that some of the numbers give different colours from obtained in MODE2. Finally, if you try using COLOUR commands in MODE0 the only colours produced will be black and white. The different modes therefore have different colour sets and colour numbers which are summarised in Fig. 3.1.

Now you will notice here that there is no mention of MODE7. The reason for this is that MODE7 has its own special commands for setting up the display colour and will not respond to the COLOUR command.

You can in fact choose any number from 0 to 127 for the text colour but all that happens with these higher numbers is that the sequence of available colours for the selected mode will be repeated. In MODE2 the colour 16 is black, the same as colour 0, 17 becomes red and so on.

An important point to note is that the COLOUR command does not affect text already displayed on the screen but only the text that is produced after the command. This means that we can have two, four or sixteen colours displayed at a time according to the display mode selected.

```
Modes 0, 3, 4 and 6 (two colour modes)

   0     Black
   1     White

Modes 1 and 5 (four colour modes)

   0     Black
   1     Red
   2     Yellow
   3     White

Mode 2  (sixteen colours)

   0     Black
   1     Red
   2     Green
   3     Yellow
   4     Blue
   5     Magenta
   6     Cyan (blue-green)
   7     White
   8     Flashing black/white
   9     Flashing red/cyan
  10     Flashing green/magenta
  11     Flashing yellow/blue
  12     Flashing blue/yellow
  13     Flashing maganta/green
  14     Flashing cyan/red
  15     Flashing white/black
```

Fig. 3.1. Text colours.

Graphics colour selection

If we select, say, yellow text and then draw some lines using either the PLOT or DRAW commands the lines will still be drawn in white because the BBC Micro has a separate colour control scheme for its graphics operations. Once again we can have the same colours as for text but we must use a different command to change the graphics colour.

The command for selecting a graphics colour is

```
GCOL N,C
```

Unlike the COLOUR command GCOL has two numbers following it. There are several modes of operation for GCOL and the first number N after the command determines how GCOL will operate.

Normally we shall use GCOLØ,C which causes the graphics to be drawn in the selected colour. Thus GCOLØ,C is equivalent to the INK command used on some other computers.

The second number after GCOL is the number of the colour to be used. This produces the same colours as we would get with the command COLOUR. Thus, in MODE1, GCOLØ,1 will produce red lines, GCOLØ,2 gives yellow lines and GCOLØ,3 draws in white.

As with text, the graphics drawing colour is automatically set to white after a mode is selected. When a new graphics colour is selected any lines already drawn are unaffected but new lines will be drawn in the selected colour.

Making patterns

Now that we can draw various kinds of shapes and control the colour of our display it becomes possible to experiment with producing patterns on the screen. We can start with a simple moving line pattern combined with colour changes.

The principle involved in drawing this type of pattern is that we start by choosing two random points (X1,Y1 and X2,Y2) on the screen and then draw a line between them. Next we alter the two points by a small amount and then draw another line. We now continue drawing lines with a small change to each end point as each new line is drawn.

If the same change in values X and Y is made at each end point the line remains the same length as it moves across the screen to produce a ribbon of colour. If different amounts of change are made at the opposite ends of the line the width of the ribbon changes and it also curls as it moves around the screen. At the edges of the screen the direction of the change can be reversed to send the ribbon back across the screen again. A pattern drawing program of this type is listed in Fig. 3.2.

Lines 14Ø–17Ø set up the initial line position and step sizes. Lines 18Ø and 19Ø clear the screen and select a random colour number. Lines 21Ø–23Ø set up the drawing colour and draw the line. Lines 24Ø–330 set up the new end points for the next line and also check for screen boundaries. At a screen boundary the X or Y step is reversed in sign and a colour change is made.

Each picture contains 1000 lines and in line 35Ø a time delay loop is included to allow the pattern to be viewed. The program produces

```
100 REM Moving line pattern
110 REM with colour changes
120 MODE2
130 X1=RND(1279):Y1=RND(1023)
140 X2=RND(1279):Y2=RND(1023)
150 FOR I = 1 TO 50
160 DX1=8-RND(16):DX2=8-RND(16)
170 DY1=8-RND(16):DY2=8-RND(16)
180 CLG
190 C% = RND(7)
200 FOR N=1 TO 1000
210 GCOL0,C%
220 MOVE X1,Y1
230 DRAW X2,Y2
240 X1=X1+DX1:Y1=Y1+DY1
250 X2=X2+DX2:Y2=Y2+DY2
260 IF X1>0 AND X1<1279 THEN 280
270 DX1=-DX1:X1=X1+DX1:C%=RND(8)-1
280 IF X2>0 AND X2<1279 THEN 300
290 DX2=-DX2:X2=X2+DX2:C%=RND(8)-1
300 IF Y1>0 AND Y1<1023 THEN 320
310 DY1=-DY1:Y1=Y1+DY1:C%=RND(8)-1
320 IF Y2>0 AND Y2<1023 THEN 340
330 DY2=-DY2:Y2=Y2+DY2:C%=RND(8)-1
340 NEXT N
350 FOR T = 1 TO 2000:NEXT T
360 NEXT I
370 END
```

Fig. 3.2 Moving coloured line program.

a succession of 50 different random patterns and the results can be quite pretty.

Producing moiré patterns

Another quite attractive type of pattern that can be produced by using high resolution graphics is the moiré pattern. It is easy to produce such patterns by simply choosing an origin point on the screen and then drawing a pattern of radial lines from that point out to the edges of the screen. According to the pitch of the lines and the position of the origins we can get a variety of very pretty patterns. Now by using different colours for alternate lines even more attractive results are achieved. The program listed in Fig. 3.3 uses random functions to generate the origins of the patterns and their pitch and also to select the colours so that a continuously changing

```
100 REM Moire patterns
110 FOR N=1 TO 100
120 MODE RND(3)-1
130 CX = RND(1100)+100
140 CY = RND(800)+100
150 ST = RND(8)+4
160 C1% = RND(8)-1
170 C2% = RND(8)-1
180 IF C1% = C2% THEN C2% = 7-C1%
190 FOR X = 0 TO 1279 STEP ST*2
200 GCOL0,C1%
210 MOVE X,0
220 DRAW CX,CY
230 DRAW 1279-X,1023
240 GCOL0,C2%
250 MOVE X+ST,0
260 DRAW CX,CY
270 DRAW 1279-X+ST,1023
280 NEXT X
290 FOR Y = 0 TO 1023 STEP ST*2
300 GCOL0,C1%
310 MOVE 0,Y
320 DRAW CX,CY
330 DRAW 1279,1023-Y
340 GCOL0,C2%
350 MOVE 0,Y+ST
360 DRAW CX,CY
370 DRAW 1279,1023-Y+ST
380 NEXT Y
390 FOR T = 1 TO 5000:NEXT T
400 NEXT N
410 END
```

Fig. 3.3. Moiré pattern program.

display is produced. Fig. 3.4 shows an example of the moire pattern display.

Lines 12Ø–18Ø set up the mode, starting position, step size and colours. Note MODE Ø, 1 or 2 is selected at random. Lines 19Ø–38Ø draw the pattern. Line 39Ø merely provides a delay to allow the pattern to be viewed.

Setting up the background colour

So far the background against which the text or graphics has been presented has always been black. The black background is automatically selected at switch-on or when the mode is changed.

By using the COLOUR and GCOL commands we can change the

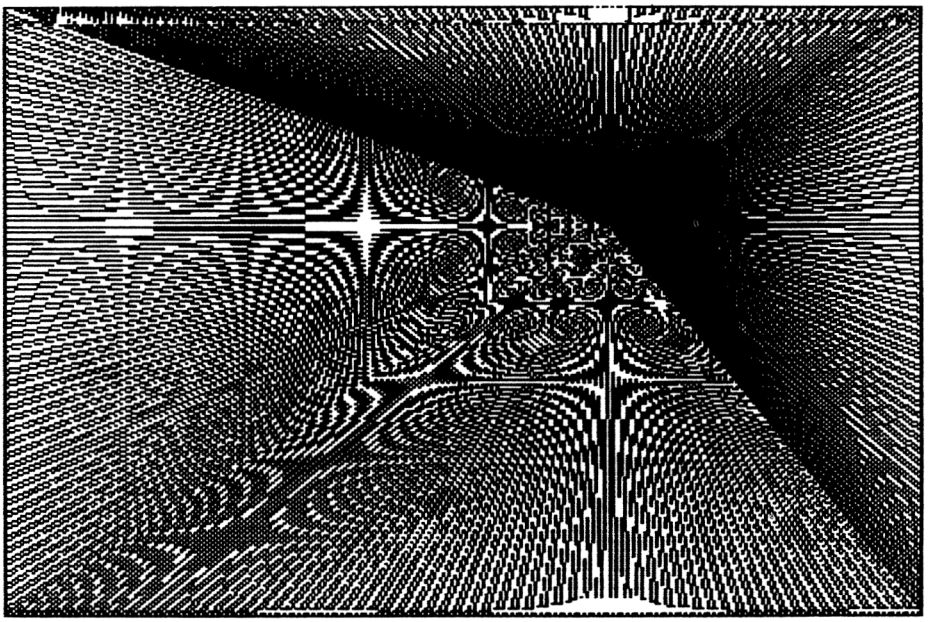


Fig. 3.4. Moiré pattern display.

background colour to any one of the colours available in the selected mode. The command format is the same as for selecting the text or graphics colour, but the colour numbers are different.

The numbers for the background colours are simply 128 higher than those for the foreground colours. Thus, in a two-colour mode the black background requires colour number 128 and a white background would be 129. In four-colour modes the background colour numbers range from 128 to 131 and for mode 2 we can use the numbers 128–143. This is summarised in Fig. 3.5.

You will very soon discover, however, that selecting the background colour by using COLOUR or GCOL will not by itself change the displayed background. This is because the background colour must be written into every pixel on the screen and this is achieved by clearing the screen. There are two commands for this purpose. One is CLS which clears the text screen, and the other is CLG which clears the graphics screen. To get the new background colour we must follow the colour selection statement with a clear screen statement. So for a red text background in mode 2 we could use

```
100 MODE 2
110 COLOUR 129
120 CLS  (clear text screen)
```

```
Two colour modes

    128 Black background
    129 White background


Four colour modes

    128 Black background
    129 Red background
    130 Yellow background
    131 White background

Mode 2

    128 Black background
    129 Red background
    130 Green background
    131 Yellow background
    132 Blue background
    133 Magenta background
    134 Cyan background
    135 White background
    136 Flashing black/white
    137 Flashing red/cyan
    138 Flashing green/magenta
    139 Flashing yellow/blue
    140 Flashing blue/yellow
    141 Flashing magenta/green
    142 Flashing cyan/red
    143 Flashing white/black
```

Fig. 3.5. Background colours.

and to set the graphics background to say blue we could add

```
130 GCOL0,131
140 CLG  (clear graphics screen)
```

As with the drawing or foreground colour we can choose to have different background colours for text and graphics. When text is written on to the graphics background each symbol space occupied by text will now have the text background colour. The graphics background will normally override the text background colour.

Filling in areas of colour

We can readily draw lines on the screen in various colours, but for

more visual impact we will often want to fill in whole areas of colour.

Suppose we decide to draw a square and then fill it in red. One possible solution would be to draw a set of horizontal lines one above the other in the square to light every pixel element, and if the colour selected is red then we have a red square. The only snag with this is that it requires a loop to draw the lines and will be fairly slow. An alternative and faster technique available on the BBC Micro is to use one of the special PLOT commands. This is

```
PLOT85,X,Y
```

or alternatively, for plotting with relative coordinates

```
PLOT81,X,Y
```

PLOT85,X,Y causes the triangle bounded by the point X,Y and the last two sets of X,Y coordinates used to be filled with the current graphics colour. Suppose we wanted to draw a square and fill it with colour. First we would use a MOVE command to set the graphics cursor at one corner of the square. Next we could draw one side using a normal DRAW or PLOT command. The second side could be drawn using either PLOT81 or PLOT85 and this will cause a half of the square to be filled with colour. The third side is drawn normally and then PLOT81 or PLOT85 is used to draw the final side and this also fills the rest of the square with colour. You could save one step by drawing first the bottom side, then across the diagonal of the square with PLOT81 or PLOT85 and finally across the top with PLOT81 or PLOT85. Try it out for yourself.

PLOT85 uses absolute X and Y coordinates, but we can also use PLOT81 where the X and Y values are relative to the current graphics cursor position. So basically to fill in a triangle we need to MOVE or DRAW to two points of the triangle and on the third point we use the colour fill PLOT81 or PLOT85 operation to fill in the area bounded by the three points. Sometimes the way in which we draw a figure may need to be changed to get the correct colour fill action.

Using the normal polygon drawing routine and PLOT85 to draw in the lines will in fact just produce a series of small filled triangles near the outer edge of the polygon, since only the first triangle will have a point at the centre of the polygon. To overcome this we need to include a PLOT statement taking the cursor to the centre of the polygon after plotting each point at the outer edge. A similar situation arises in the case of a circle or an ellipse and here again a

MOVE or PLOT to the centre of the figure after each point on the circumference will ensure that the whole circle is filled. If we change the colour whilst drawing the circle then a segment of the circle will be in one colour and the rest in another. So when using colour fill the actual drawing sequence may need to be altered to get the desired results.

Patterns using colour fill

In Fig. 3.6 a listing is given for a program to draw a series of varying sized squares at random points on the screen and to fill the squares with random colours. Mode 2 is used but the range of colours has been limited to colour numbers 0–7 which are all steady colours.

Lines 13Ø–15Ø set up the random square position, size and colour. Lines 16Ø–21Ø draw and fill the square.

```
100 REM Random coloured squares.
110 MODE 2
120 FOR N= 1 TO 150
130 X=RND(1200):Y=RND(1000)
140 W=40 + RND(200)
150 C%=RND(8)-1
160 MOVE X,Y
170 GCOL0,C%
180 PLOT1,W,0
190 PLOT81,0,W
200 PLOT1,-W,0
210 PLOT81,0,-W
220 NEXT N
230 END
```

Fig. 3.6. Program for producing colour filled squares.

Mirror image patterns

By reversing the signs of the X and Y coordinates and plotting these relative to a point at the centre of the screen we can produce patterns consisting of four mirror images in each of the quarters of the screen around the middle point. The effect is similar to that produced by a kaleidoscope and produces very attractive patterns if we use the colour fill method of plotting.

A program to produce a series of kaleidoscope style patterns is given in Fig. 3.7. Here two random X,Y points are chosen and

```
100 REM Kaleidoscope patterns
110 MODE 2
120 CX=640:CY=512
130 FOR N=1 TO 20
140 X1=RND(500):Y1=RND(400)
150 X2=RND(500):Y2=RND(400)
160 C%=RND(8)-1
170 GCOL0,C%
180 MOVE CX,CY
190 DRAW CX+X1,CY+Y1
200 PLOT85,CX+X2,CY+Y2
210 MOVE CX,CY
220 DRAW CX-X1,CY+Y1
230 PLOT85,CX-X2,CY+Y2
240 MOVE CX,CY
250 DRAW CX+X1,CY-Y1
260 PLOT85,CX+X2,CY-Y2
270 MOVE CX,CY
280 DRAW CX-X1,CY-Y1
290 PLOT85,CX-X2,CY-Y2
300 NEXT N
310 FOR T=1 TO 5000: NEXT T
320 CLG
330 GOTO 130
```

Fig. 3.7. Kaleidoscope pattern program.

plotted at four different positions relative to the centre of the screen (CX,CY). This is achieved by using the four possible combinations of signs on the values of X and Y before they are added to the coordinates CX,CY and the points are plotted. A random colour change has also been included to provide colourful patterns which will continue to be drawn indefinitely. Press ESCAPE to halt the program.

Lines 14Ø–17Ø set up the random points and colour. Lines 18Ø–3ØØ draw one segment of the complete pattern. Line 31Ø provides a delay to allow the pattern to be viewed before the screen is cleared by line 32Ø ready for the next pattern.

Windows

There will be occasions when we want to reserve a part of the screen for text and use the rest for graphics. An example of this might be in say an adventure game where a graphics picture is used to show the current location but we also need a separate area of screen on which the commands, lists of objects being carried or other useful information might be displayed. On the BBC Micro it is quite easy to

display text on a graphics image by simply using PRINT commands combined with the TAB function to place the text in a particular position. A more convenient technique in some cases is to make use of text and graphics 'windows'. These windows are areas of the screen which may be set aside for the display of text or graphics as required.

Making a text window

Let us suppose we want to reserve an area of the screen for text only. We can do this by using the VDU28 command which has two sets of X,Y coordinates, following it as follows:

```
VDU28 X1,Y1,X2,Y2
```

where X1,Y1 are the coordinates of the bottom left corner of the area we want to be a window, and X2,Y2 are the coordinates of the top right corner. Note that here the coordinates used are those for text symbols and will vary according to the display mode in use.

When the window is set up we can set the text background colour by using COLOUR and CLS, but instead of filling the screen it will now only affect the text window area. If we now set up a graphics background colour it will change the background over the rest of the screen but will not affect the text window area.

Making a graphics window

In the same way that a text window area can be set up we can also set aside part of the screen for exclusive use of graphics. In this case we use the command VDU24 with two sets of graphics coordinates. The command has the format

```
VDU24,X1;Y1;X2;Y2;
```

and as for text X1,Y1 is the bottom left corner and X2,Y2 is the top right corner of the required window area. In the case of graphics the coordinates are independent of the mode in use, provided of course that it is a graphics mode. Note that there is no graphics window in the teletext mode.

Now we can have two different background colours. Normally when the whole screen is used the background colour will be either a text or graphics colour according to which was the last to be set up, since both text and graphics clear commands will clear the whole

screen. When windows have been set up the CLS command will affect only the text areas whilst CLG will affect only the areas set aside for graphics.

Chapter Four
Graphs and Charts

By using a computer we can readily carry out lots of measurements or calculations and end up with enormous arrays of numbers. Having produced all of these numbers one method of presenting them is to produce a list or perhaps a table of figures. Unfortunately such a table or list of numbers is not particularly helpful when we come to interpret the results. A much better method of displaying results is to show them visually using a graphics display or perhaps a chart. One of the simplest examples of this is a thermometer type of display as shown in Fig. 4.1.

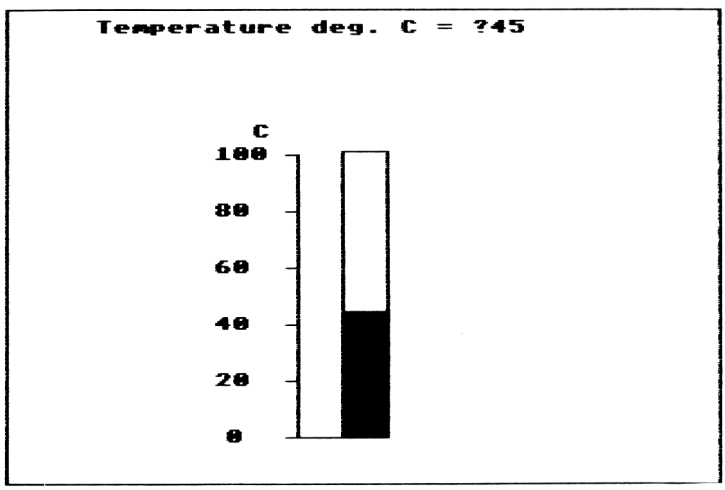


Fig. 4.1. Thermometer display.

Thermometer display

In a conventional mercury thermometer the length of the column of mercury indicates the temperature. In Fig. 4.1 a narrow vertical

rectangular box is drawn to represent the body of the thermometer. Inside the box is a solid vertical bar which like the mercury in the thermometer will vary in length as the quantity we are measuring changes. It is convenient to use a different colour, such as red, for the bar representing the quantity being measured. For the moment let us assume that we are measuring temperature.

As each new value of temperature is entered into the computer the red line moves up or down proportionately. In order to understand what the line means we need a scale drawn alongside the thermometer display. The scale and the outline box need be drawn only once at the start of the program but the bar is redrawn for each new reading to be displayed.

To mark up the values on the scale we shall need to write text on to the graphics display at the appropriate positions. This could be done by using the normal text cursor and a TAB instruction in the PRINT statement. Unfortunately this involves a little calculation to sort out a suitable cursor position for the text and often it is not possible to place the text in exactly the right place because it falls between two text cursor spaces. On the BBC Micro there is a very useful feature which allows the text and graphics cursors to be linked. Now the first text symbol will be printed at the position of the graphics cursor. Here we need to note that the top left-hand corner of the character is positioned at the graphics cursor position. Knowing the size of the text symbol it is now possible to place the text where you want it. The command to link the cursors is VDU5 and to restore normal text operation VDU4 may be used.

Text symbols are 32 graphics units high and their width depends upon the mode selected. Mode 0 gives a width of 16 units, mode 1 gives 32 units and mode 2 gives 64 units. When text is printed after using VDU5 its colour will be the graphics colour and not the text colour currently in use, so GCOL is needed if you want to change the colour of linked text.

A program to produce a thermometer style display is given in Fig. 4.2. Lines 13Ø–18Ø draw the thermometer outline. Lines 2ØØ–34Ø draw in the scale. Note VDU5 links text and graphics cursors so that text is printed at the graphics cursor position.

Line 29Ø moves the cursor to the position where we want to write text for the scale. In mode 1 each symbol is 32 units square and the symbol is drawn with the graphics cursor at its top left corner. To calculate the shift needed in line 29Ø four symbol spaces are allowed for. Thus the cursor is shifted left by 128. To align the middle of the symbol with the scale mark the cursor is moved up half a symbol

```
100 REM Thermometer display
110 MODE1
120 CX=600:CY=100
130 GCOL0,3
140 MOVE CX,CY
150 PLOT1,80,0
160 PLOT1,0,610
170 PLOT1,-80,0
180 PLOT1,0,-610
190 REM Draw scale
200 VDU5
210 PLOT1,-100,0
220 PLOT0,20,0
230 PLOT0,-128,16
240 PRINT "0";
250 MOVE CX-80,CY
260 FOR T=20 TO 100 STEP 20
270 DRAW CX-80,CY+T*6
280 PLOT1,-20,0
290 PLOT0,-128,16
300 PRINT STR$(T);
310 MOVE CX-80,CY+T*6
320 NEXT T
330 PLOT0,-80,64
340 PRINT "C";
350 VDU4
360 CX=CX+4:CY=CY+4
370 REM Display temperature
380 INPUT TAB(0,0) "Temperature deg.
    C = ",T
390 IF T<100 AND T>0 THEN 420
400 PRINT TAB(0,0) "TRY AGAIN";CHR$
    (7)
410 FOR D=1 TO 500:NEXT:GOTO 380
420 MOVE CX,CY
430 GCOL0,1
440 PLOT1,72,0
450 PLOT85,CX,CY+T*6
460 PLOT81,72,0
470 GCOL0,0
480 PLOT85,CX,CY+600
490 PLOT81,72,0
500 PRINT TAB(0,0):GOTO 380
510 END
```

Fig. 4.2. Program to draw a thermometer display.

space, which is 16 units. For other modes the symbol space size can be calculated as 1280/N where N is the number of symbols per row for the selected mode.

Note that an absolute move is used after printing the text. This is because there may be two or three text symbols printed and you cannot be sure of the final graphics cursor position after the symbols are printed.

Line 35ϕ restores the normal text cursor. Lines 43ϕ–48ϕ produce the red bar and blank out the area above the new red bar.

In this particular routine the values for temperature are entered from the keyboard in response to a message at the top of the screen. Of course you could connect the computer to an electronic thermometer and input actual temperature data via the analog input channel, then the screen display would act as a real thermometer.

All kinds of information can be displayed using this style of presentation and it will often be found in games where perhaps the energy or fuel level of a spaceship or the speed of a car could be displayed. The thermometer display could be arranged horizontally, much like the moving strip speedometers fitted to some cars, and this is achieved by simply exchanging X and Y in the DRAW and PLOT statements for the thermometer part of the display. Remember of course that the scale also needs to be drawn horizontally. Try it as an exercise.

Bar charts

Whilst the thermometer style display is useful to show the current state of some measurement we will often want to see how the

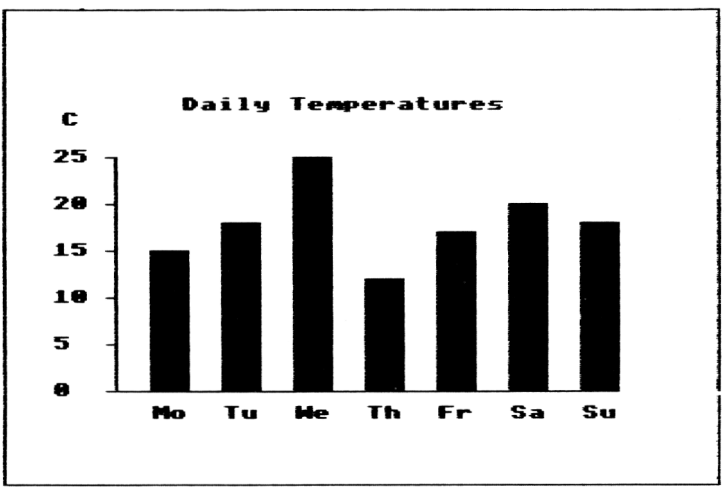


Fig. 4.3. Temperature bar chart.

```
100 REM Simple bar chart
110 MODE1
120 DIM DAY$(7),T(7)
130 DATA "Mo",15,"Tu",18,"We",25,
   "Th,12,"Fr",17,"Sa",20,"Su",18
140 FOR N = 1 TO 7
150 READ DAY$(N),T(N)
160 NEXT N
170 CX=200:CY=200
180 VDU5
190 GCOL0,3
200 REM Draw axes and scales
210 MOVE CX,CY
220 DRAW CX+28*32,CY
230 FOR N = 1 TO 7
240 MOVE CX+N*128-64,CY-32
250 PRINT DAY$(N);
260 NEXT N
270 MOVE CX,CY
280 PLOT1,-16,0
290 PLOT0,-96,16
300 PRINT "0";
310 MOVE CX,CY
320 FOR C=5 TO 25 STEP 5
330 DRAW CX,CY+C*20
340 PLOT1,-16,0
350 PLOT0,-96,16
360 PRINT STR$(C);
370 MOVE CX,CY+C*20
380 NEXTC
390 PLOT0,-96,96
400 PRINT "C";
410 MOVE CX,CY
420 REM Plot chart
430 CX=CX-64
440 GCOL0,1
450 FOR N=1 TO 7
460 MOVE CX+128*N,CY
470 PLOT1,64,0
480 PLOT85,CX+128*N,CY+T(N)*20
490 PLOT81,64,0
500 NEXT N
510 VDU4
520 PRINT TAB(10,6);"Daily Temperatu
    res";
530 END
```

Fig. 4.4. Program to draw a simple bar chart.

situation varied over a period of time. We could perhaps measure the temperature at noon on each day of the week. This can easily be

arranged by drawing the thermometer displays for the days of the week alongside one another as shown in Fig. 4.3 In this case only the variable length bar is drawn and one common scale is included at the left-hand side. To improve visibility the bars may be drawn with a slight gap between adjacent bars. This type of display is called a *bar graph* or more commonly a *bar chart*. The program for drawing a simple bar chart showing the daily temperatures over a period of a week is listed in Fig. 4.4.

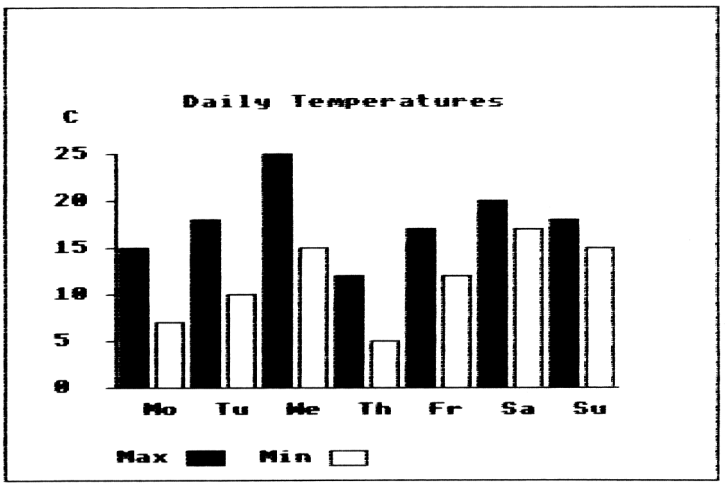


Fig. 4.5. Multiple bar chart.

Lines 2ϕϕ–4ϕϕ draw in the scales. Lines 43ϕ–52ϕ draw the bars and print the title heading for the chart.

Bar charts are frequently used in business applications to show the trend in sales over a year, or perhaps the stock level, number of orders or income over a period. It is very easy to see the trend of the results on such a chart. Sometimes the colour of the bar is changed if its level goes above or below a specified limit. This can provide an easily recognised warning that a situation is becoming dangerous or needs attention. In such cases either the whole bar changes colour or the part above the limit line might change colour. Lines 13ϕ–16ϕ set up the data. Data could alternatively be keyed in or read from a data file by altering this part of the program.

Multiple bar charts

When two different variables are to be displayed on the same chart

the bars are drawn in pairs so that they become interleaved, as shown in Fig. 4.5. To provide clearer distinction between the sets of bars a different colour is used for each set of bars. Three or four graphs could be interleaved in this way if desired. A typical application for a multiple bar chart might show the income and expenditure on a single chart. It might also be useful to show perhaps the predicted income and expenditure as well as to see how

```
100 REM Multiple bar chart
110 MODE1
120 DIM DAY$(7),T1(7),T2(7)
130 DATA "Mo",15,7,"Tu",18,10,"We",
    25,15,"Th",12,5,"Fr",17,2,"Sa",
    20,7,"Su",18,15
140 FOR N = 1 TO 7
150 READ DAY$(N),T1(N),T2(N)
160 NEXT N
170 CX=200:CY=200
180 VDU5
190 GCOL0,3
200 REM Draw axes and scales
210 MOVE CX,CY
220 DRAW CX+28*32,CY
230 FOR N = 1 TO 7
240 MOVE CX+N*128-96,CY-32
250 PRINT DAY$(N);
260 NEXT N
270 MOVE CX,CY
280 PLOT1,-16,0
290 PLOT0,-96,16
300 PRINT "0";
310 MOVE CX,CY
320 FOR C=5 TO 25 STEP 5
330 DRAW CX,CY+C*20
340 PLOT1,-16,0
350 PLOT0,-96,16
360 PRINT STR$(C);
370 MOVE CX,CY+C*20
380 NEXTC
390 PLOT0,-96,96
400 PRINT "C";
410 MOVE CX,CY
420 REM Plot chart
430 FOR N=1 TO 7
440 GCOL0,1
450 PLOT0,8,0
460 PLOT1,48,0
470 PLOT81,-48,T(N)*20
```

```
480 PLOT81,48,0
490 GCOL0,2
500 PLOT0,16,-T1(N)*20
510 PLOT1,48,
530 PLOT81,-48,T2(N)*20
540 PLOT81,48,0
550 PLOT0,8,-T2(N)*20
560 NEXT N
570 MOVE 328,840
580 GCOL0,3
590 PRINT"Daily Temperatures";
600 MOVE 200,72
610 GCOL0,1
620 PRINT"Max";
630 PLOT0,64,0
640 PLOT1,0,-32
650 PLOT81,80,0
660 PLOT81,0,32
670 PLOT0,64,0
680 GCOL0,2
690 PRINT"Min";
700 PLOT0,64,0
710 PLOT1,0,-32
720 PLOT81,64,0
730 PLOT81,0,32
740 END
```

Fig. 4.6. Program to produce a multiple bar chart with interleaved bars.

the actual values compare with predicted trends. Bar graphs are often used in financial and production charts since they provide a bolder and easier to follow presentation than a list of figures.

As an example of a multiple bar chart see the program listed in Fig. 4.6, which produces a plot of the maximum and minimum temperatures for the days of a week.

Pie charts

A rather attractive form of display chart frequently used in business is the *pie chart*. This is used to show the proportions into which something divides up. An example might be the percentage votes for political parties derived from a poll or sample of electors. We have all seen these charts displayed on television. Another application might be to show how the resources of a company are used or how its money has been spent.

As its name implies the pie chart is effectively like a plan view of a

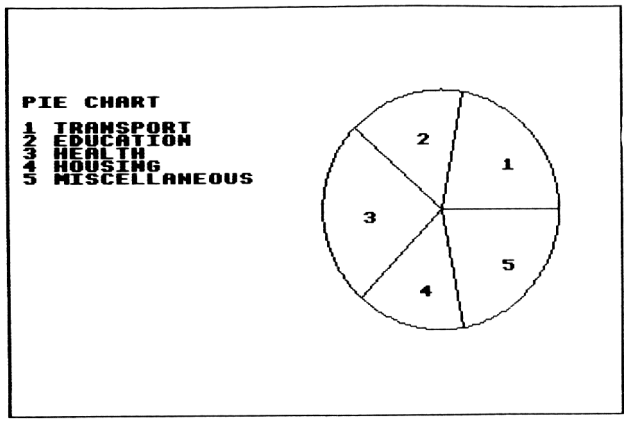


Fig. 4.7 A typical pie chart display.

pie which has been sliced up into segments of various sizes. Each slice of the pie represents one item and shows the percentage of the total made up by that item. A typical pie chart is shown in Fig. 4.7.

To draw a pie chart we are effectively drawing a series of segments of a circle. The angle for each segment can be calculated as a percentage of 2 × pi (360 degrees). Conversion to X and Y values for plotting is carried out in much the same way as we used to draw a circle using trigonometric methods. In fact we are really drawing arcs of a circle and then at the end of each arc we draw a radial line into the centre of the chart. A simple pie chart drawing program, allowing up to five segments, is shown in Fig. 4.8.

Lines 13ø–16ø set up the data for the segments. Line 17ø moves the graphics origin to the centre of the screen. Line 21ø links text and graphics cursors to allow text to be placed where we want it. Lines 22ø–33ø draw the chart. Note that in lines 28ø–3øø a point is chosen halfway through the segment and inside the edge of the chart to allow the item number to be printed. Lines 36ø–42ø print the heading and the key to the segment items.

Coloured pie charts

For best effect, pie charts are usually drawn with each segment filled in a different colour. We can do this by using PLOT85 and moving back to the centre between plotting each point around the circle. This fills in a very narrow triangular slice of the circle at each step.

Item numbers can be written into each segment after the chart has been drawn by linking the text and graphics cursors using VDU5,

```
100 REM Simple Pie Chart
110 MODE1
120 DIM S(5)
130 DATA 22,16,25,15,22
140 FOR N = 1 TO 5
150 READ S(N)
160 NEXT N
170 VDU29,900;512;
180 R=300:THS=0:DTH=2*PI/300
190 MOVE 0,0
200 REM Plot chart
210 VDU5
220 FOR N=1 TO 5
230 ANG=2*PI*S(N)/100
240 DRAW R*COS(THS),R*SIN(THS)
250 FOR TH=THS TO THS+ANG STEP DTH
260 DRAW R*COS(TH),R*SIN(TH)
270 NEXT TH
280 THT=THS+ANG/2:RT=200
290 MOVE RT*COS(THT),RT*SIN(THT)
300 PRINT STR$(N);
310 MOVE 0,0
320 THS=THS+ANG
330 NEXT N
340 VDU4
350 VDU29,0;0;
360 PRINT TAB(0,7) "Pie Chart"
370 PRINT
380 PRINT "1  Transport"
390 PRINT "2  Education"
400 PRINT "3  Health"
410 PRINT "4  Housing"
420 PRINT "5  Miscellaneous"
430 END
```

Fig. 4.8. Program to draw a simple pie chart.

then printing the appropriate text in the segment. The allocation or meaning of each segment can either be indicated simply by colour and by having a colour key alongside the chart, or by using segment numbers as in the simple chart.

Scientific graphs

Although the bar chart and pie chart are well suited to business use, when we come to scientific or mathematical graph plotting a slightly different arrangement is used.

The layout is similar to that of a bar chart, with the results of the

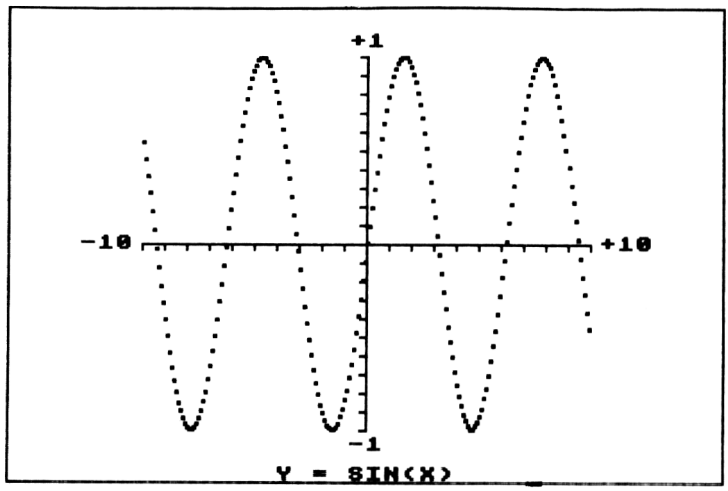


Fig. 4.9. Typical scientific type graph showing Y = SIN(X).

calculation or experiment plotted vertically on the screen and the measurement steps horizontally. Instead of drawing a bar, the value of Y is simply shown as a dot at a point equivalent to the top of the bar on a bar chart. Sometimes to make the point easier to see a small + sign, triangle or circle may be used as a marker instead.

In a bar chart the variables are normally positive but in a scientific graph the variables X and Y may be either positive or negative. To cater for this the X and Y axes are drawn as shown in Fig. 4.9. Positive values of X are to the right of the vertical Y axis and negative values of X to the left. Similarly positive Y is above the X axis and negative below.

As an example let us draw the graph for Y = sin(X) with values of X from −10 to +10. Firstly the value of sin(X) will vary between −1 and +1 so this determines the scale of the Y axis. To produce a reasonable size graph we shall arrange that 10 units of X and 1 unit of Y will be 400 units on the screen, so we set up two scale factors SX and SY. On the scales we shall show 10 divisions each side of zero. The program listing is given in Fig. 4.10.

Lines 17Ø–32Ø draw the X axis. Lines 33Ø–48Ø draw the Y axis. Lines 5ØØ–54Ø calculates Y and plot the graph. The result is as shown in Fig. 4.9, with a white set of axes and yellow dots for the plotted points.

Interpolation

In plotting the sine function we used a very large number of values so

```
100 REM Sine graph
110 MODE1
120 SX=40:SY=400
130 VDU29,640;512;
140 VDU5
150 MOVE0,0
160 REM Draw scales
170 FOR X=-1 TO -10 STEP -1
180 DRAW X*SX,0
190 DRAW X*SX,-10
200 MOVE X*SX,0
210 NEXT X
220 PLOT0,-112,16
230 PRINT "-10";
240 MOVE0,0
250 FOR X=1 TO 10
260 DRAW X*SX,0
270 DRAW X*SX,-10
280 MOVE X*SX,0
290 NEXT X
300 PLOT0,16,16
310 PRINT "+10";
320 MOVE0,0
330 FOR Y=-0.1 TO -1.1 STEP -0.1
340 DRAW 0,Y*SY
350 DRAW -10,Y*SY
360 MOVE 0,Y*SY
370 NEXT Y
380 PLOT0,-32,-16
390 PRINT "-1";
400 MOVE0,0
410 FOR Y=0.1 TO 1.1 STEP 0.1
420 DRAW 0,Y*SY
430 DRAW -10,Y*SY
440 MOVE 0,Y*SY
450 NEXT Y
460 PLOT0,-32,48
470 PRINT "+1";
480 MOVE0,0
490 REM Plot graph
500 GCOL0,2
510 FOR X=-10 TO +10 STEP 0.1
520 Y=SIN(X)
530 PLOT69,X*SX,Y*SY
540 NEXT X
550 PLOT4,-160,-480
560 PRINT "Y = SIN(X)";
570 VDU4
580 END
```

Fig. 4.10. Program to draw sine graph.

that the points were closely spaced and gave a good picture of the shape of the curve produced by the function. If there were less values for X and Y the points would tend to be spread apart, giving a less clear impression of the function shape.

Sometimes we may wish to find the probable value for Y at a value of X that was not included in the points used for the graph. By using a technique known as *interpolation* we can obtain an approximate value for such an intermediate point on the curve.

The simplest technique for interpolation is to join successive points on the curve by straight lines. This is generally known as *linear interpolation.* We can in fact join the points with a straight line as the graph is plotted. This gives an easier to follow curve when the number of points available is limited. Some care is needed, however, because unless there are a reasonable number of points the straight line interpolation technique can be wildly inaccurate.

To join the points the graph plotting routine is simply altered to use a DRAW command instead of the PLOT69 command. A point to remember here is that you will need to make a MOVE operation to get to the first plotted point, otherwise a line will be drawn across the graph. This usually means that you have to calculate or obtain the first X and Y values before going into the main plotting loop.

There are more advanced techniques for interpolation which involve fitting a mathematical curve through each three successive points, but it is felt that these are beyond the scope of this book. For most applications the linear interpolation technique is satisfactory provided there are enough points on the graph to start with.

Dial and clock type displays

For some applications a dial and pointer or clock type of display may be required. Typical uses might be in the instrument panel for a flight simulator program or perhaps to provide an instrument readout for an experiment where the computer is monitoring the results. In some cases, of course, the display may show time elapsed or time remaining in a game program.

The basic display consists of a circular, or possibly polygon shaped, dial with either one or two pointers. The pointers may simply be radial lines or a filled shape drawn radially from the centre of the dial. The dial itself may also be filled with colour to make it stand out from the background. A scale of some sort is usually

drawn around the outside of the dial.

This analogue type of display is often much more convenient where precise readings are not required but where the general trend can be taken in at a glance. An example of this is in digital and analogue watches and clocks. Although the digital display is precise it is much easier to tell the time by just glancing at a conventional clock face.

Drawing the dial is quite straightforward since it just involves drawing and perhaps filling a circle, which may be achieved by using one of the methods described in Chapter 2. The trigonometric or rotation methods are usually best for this purpose, especially if the dial is to be filled with colour.

An important point about the hand or pointer is that it normally rotates clockwise for increasing readings. The position of the hand is readily calculated by using the rotation equations. The angle of rotation required is simply the ratio of the scale reading to full scale multiplied by the total angle represented by full scale. If all 360 degrees are used, as in a compass display, then the angle TH is given by

```
TH = 2*PI*X/FS
```

Where X is the measured value, FS is the full scale reading and TH is the angle of rotation. Because the rotation equations assume that the angle increases in an anticlockwise direction the sign of TH should be changed before inserting it into the equations. An angle of −TH is effectively the same as a clockwise rotation of TH.

Sometimes the dial may cover only 90, 180 or 270 degrees. In this case the 2PI term in the above equation should be reduced to the desired full scale angle measured in radians. The angle in radians is easily found by using the following equation:

```
RAD = DEG * PI / 18Ø
```

Normally the rotation equation assumes that the zero point is horizontal and to the right. If you want zero to be at the top as in a compass (i.e. true North = 0) then 90 degrees or PI/2 must be added to values of TH before the values of X and Y are calculated. Note that in this case a simplified rotation equation is used where the old value of X is assumed to be 0.

Drawing the scale marks is really similar to drawing the pointer except that the start of the mark line is at some radius a bit larger than the dial circle. The inner and outer ends of the mark are calculated using the rotation equation with two different values for

R. The centre point of any text used for scale calibration can be calculated in the same way using a radius larger than the outer radius of the scale marks. Remember that having found the centre point for the text an offset has to be calculated to place the cursor at the top left corner of the first symbol, using the fact that symbols are 32 units high and either 16, 32 or 64 units wide according to the display mode used.

Usually the pointer will have to be redrawn for each new reading and the old pointer mark must be erased by redrawing it in the same colour as the dial fill, or the background if the dial is not filled with colour. A procedure can be used to erase and redraw the pointer each time a new reading is calculated.

The program listed in Fig. 4.11 produces a simple dial display with a single pointer and gives a display similar to that shown in Fig. 4.12. In this program the dial starts with the pointer pointing up and has a 270 degree scale.

```
100 REM Dial and pointer display
110 MODE1
120 GCOL0,2
130 CX=640:CY=512
140 RD=240:RS1=256:RS2=296:RT=360
150 RP=220:LP=0
160 PZ=PI/2:FS=3*PI/2
170 VDU5
180 PROCDIAL
190 GCOL0,3
200 PROCSCALE
210 FOR K= 1 TO 6
220 FOR N= 1 TO 8 STEP 0.1
230 PROCPOINT
240 FOR T=1 TO 100:NEXT T
250 NEXT N
260 NEXT K
270 VDU4
280 END

400 DEFPROCDIAL
410 TH=0:DTH=2*PI/RD
420 FOR J=1 TO RD
430 MOVE CX+RD*COS(TH),CY+RD*SIN(TH)
440 TH=TH+DTH
450 DRAW CX+RD*COS(TH),CY+RD*SIN(TH)
460 PLOT85,CX,CY
470 NEXT J
480 ENDPROC
```

```
500 DEFPROCSCALE
510 TH=PZ:DTH=-2*PI/RS1
520 NP=INT(FS*RS1/(2*PI))
530 FOR J=1 TO NP
540 MOVE CX+RS1*COS(TH),CY+RS1*SIN(TH)
550 TH=TH+DTH
560 DRAW CX+RS1*COS(TH),CY+RS1*SIN(TH)
570 NEXT J
580 FOR N=0 TO 8
590 TH=PZ-FS*N/8
600 MOVE CX+RS1*COS(TH),CY+RS1*SIN(TH)
610 DRAW CX+RS2*COS(TH),CY+RS2*SIN(TH)
620 MOVE CX+RT*COS(TH),CY+RT*SIN(TH)
630 PLOT0,-16,16
640 PRINT STR$(N);
650 NEXT N
660 ENDPROC

800 DEFPROCPOINT
810 TH=PZ-FS*LP/8
820 MOVE CX,CY
830 GCOL0,2
840 DRAW CX+RP*COS(TH),CY+RP*SIN(TH)
850 TH=PZ-FS*N/8
860 MOVE CX,CY
870 GCOL0,1
880 DRAW CX+RP*COS(TH),CY+RP*SIN(TH)
890 LP=N
900 ENDPROC
```

Fig. 4.11. Program to draw a dial and pointer display.

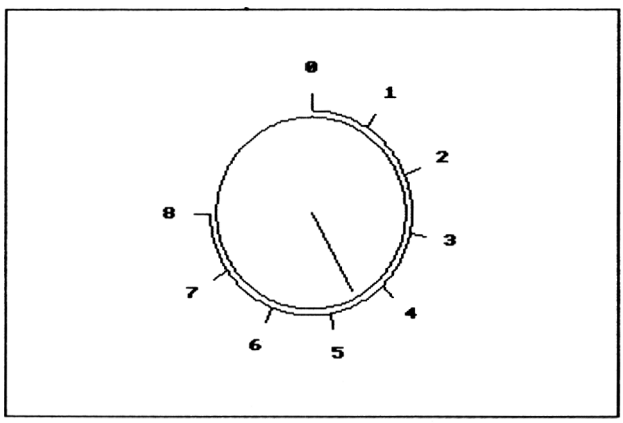


Fig. 4.12. Dial and pointer type display.

If two hands are required, as in a conventional clock display, then the same basic drawing routine may be used but with different radius for each hand. Equally well a third pointer or hand might be added. If the pointers are to have different shapes it may be convenient to have a separate drawing procedure for each pointer.

Moving sector displays

A variation on the dial type display is the moving sector type, where a sector of the dial is filled in a different colour from the rest of the dial. As the reading increases the moving sector takes up a greater part of the complete circle.

The basic principles for generating this type of display are a little different to those of a pointer and dial type. Here the angular value

```
100 REM Moving sector display
110 MODE1
120 GCOL0,2
130 CX=640:CY=512
140 RD=240:RS1=256:RS2=296:RT=360
150 RP=220:LP=0
160 PI=PI/2:FS=3*PI/2
170 VDU5
180 PROCDIAL
190 GCOL0,3
200 PROCSCALE
210 FOR K= 1 TO 6
220 FOR N= 1 TO 8
230 PROCSECTOR
240 NEXT N
250 FOR N= 8 TO 0 STEP -1
260 PROCSECTOR
270 NEXT N
280 NEXT K
290 VDU4
300 END

400 DEFPROCDIAL
410 TH=0:DTH=2*PI/RD
420 FOR J=1 TO RD
430 MOVE CX+RD*COS(TH),CY+RD*SIN(TH)
440 TH=TH+DTH
450 DRAW CX+RD*COS(TH),CY+RD*SIN(TH)
460 PLOT85,CX,CY
470 NEXT J
480 ENDPROC
```

```
500 DEFPROCSCALE
510 TH=PZ:DTH=-2*PI/RS1
520 NP=INT(FS*RS1/(2*PI))
530 FOR J=1 TO NP
540 MOVE CX+RS1*COS(TH),CY+RS1*SIN(TH)
550 TH=TH+DTH
560 DRAW CX+RS1*COS(TH),CY+RS1*SIN(TH)
570 NEXT J
580 FOR N=0 TO 8
590 TH=PZ-FS*N/8
600 MOVE CX+RS1*COS(TH),CY+RS1*SIN(TH)
610 DRAW CX+RS2*COS(TH),CY+RS2*SIN(TH)
620 MOVE CX+RT*COS(TH),CY+RT*SIN(TH)
630 PLOT0,-16,16
640 PRINT STR$(N);
650 NEXT N
660 ENDPROC

800 DEFPROCSECTOR
810 TH1=PZ-FS*LP/8
820 TH2=PZ-FS*N/8
830 DTH=-2*PI/RP
840 NP=ABS(INT((TH2-TH1)/DTH))
850 GCOL0,1
860 IF NP=0 THEN 950
870 IF TH1<TH2 THEN DTH=-DTH:GCOL0,2
880 TH=TH1
890 FOR J=1 TO NP
900 MOVE CX+RP*COS(TH),CY+RP*SIN(TH)
910 TH=TH+DTH
920 DRAW CX+RP*COS(TH),CY+RP*SIN(TH)
930 PLOT85,CX,CY
940 NEXT J
950 LP=N
960 ENDPROC
```

Fig. 4.13. Program to draw a moving sector display.

of the moving sector is calculated in the same way as for calculating pointer position. The actual display is then drawn by using a variant of the circle filling routine which draws and fills only part of the circle. The colour is then changed and the rest of the circle is filled.

Scale markings and calibrations can be drawn in the same way as for a dial and pointer type of display. It is usually easiest to redraw the whole dial for each new sector position but a faster technique might be to compare the new and current sector values and just draw or erase the small sector representing the difference. If the new value is larger than the old the program draws an additional piece of sector, moving clockwise and using the sector colour. If the new

value is smaller then a small sector equal to the difference is drawn anticlockwise in the basic dial colour.

Fig. 4.13 shows a simple program for drawing this type of display and the results on the screen will be similar to Fig. 4.14.

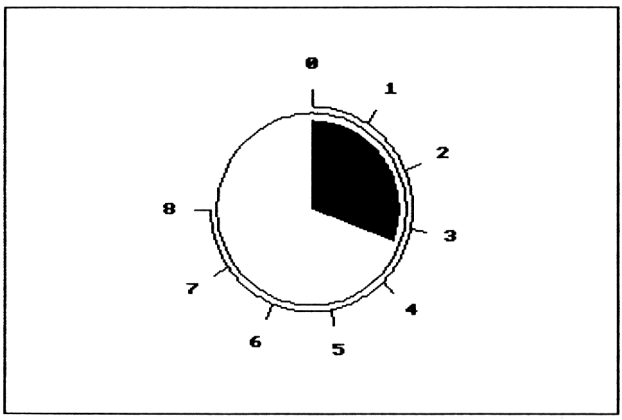


Fig. 4.14. A moving sector type display.

Chapter Five

More About Colour and Plot

So far we have looked at the basic arrangements for selecting colour and for plotting or drawing on the BBC Micro. There are, however, several more possibilities in the colour control system and in the use of the PLOT command which we shall now explore.

Changing the palette

When an artist is painting a picture he does not take all of the paints out of his paint box but selects a few colours that he needs for the picture. These are placed on that strangely shaped board which he calls his 'palette'. In the BBC Micro we have a similar situation, especially in the two- and four-colour modes. Normally the colours are set up to the combinations we have been using and this is what might be called the 'standard' or default palette. By using some special VDU commands we can in fact change the set of colours available in the two- and four-colour modes. In mode 2 we can rearrange the colours in a different order or have the same colour displayed for several different colour numbers.

At this point we need to look at the colour numbering in a slightly different way. The numbers that we use in the COLOUR and GCOL commands to specify colour are what are known as 'logical' colour numbers. You will recall that in mode 1 the actual colours produced by colour numbers 2 and 3 are different from those produced by the same numbers when mode 2 is selected. This is because the display colour assigned to these logical colour numbers is different in these two modes.

The colour that is actually displayed on the screen in response to a colour command is called the 'actual' colour number. These actual colour numbers are listed in fig. 5.1. You will notice that the numbers and colours are the normal ones that apply for mode 2.

```
Colour 0    Black
Colour 1    Red
Colour 2    Green
Colour 3    Yellow
Colour 4    Blue
Colour 5    Magenta
Colour 6    Cyan
Colour 7    White
Colour 8    Flashing Black/white
Colour 9    Flashing Red/cyan
Colour 10   Flashing Green/magenta
Colour 11   Flashing Yellow/blue
Colour 12   Flashing Blue/yellow
Colour 13   Flashing Magenta/green
Colour 14   Flashing Cyan/red
Colour 15   Flashing white/black
```

Fig. 5.1. Actual colour numbers.

All of the colours produced are combinations of the primary colours red, green and blue and if we examine the numbering scheme we shall see that for the foreground or ink colour only the least significant four bits of the number are used to define the colour. One bit in the number is assigned to each of the colours red, green and blue, whilst a fourth bit indicates whether the colour is flashing or not. The bit assignment is shown in Fig. 5.2. Thus for yellow, which is a combination of red and green, we find that bits 1 and 2 are both at 1, giving the number 3 and the colour R+G = Yellow.

In the BBC Micro there is a small section of memory set aside to hold the set of actual colour numbers which make up the current palette. As each pixel is scanned on the screen the logical colour number is used to select one of the locations in this palette memory and the corresponding actual colour number is read out. This colour number determines the combination of red, green and blue that is

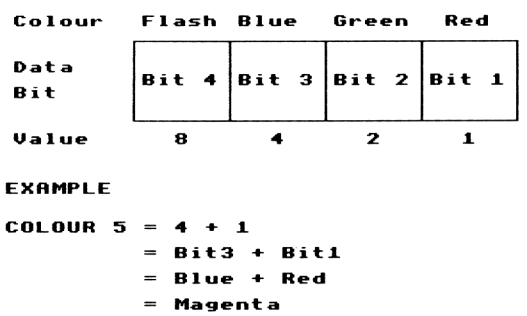


Fig. 5.2. Assignment of bits and numerical values for actual colour numbers.

displayed on the screen for that pixel.

By using a VDU command we can in fact assign any of the actual display colours to a logical colour number. Thus if we wanted to have green text or graphics in mode 0 we would assign the actual colour number 2 (green) to logical colour 1. In effect we have taken a different colour paint out of the box and put it on our brush. In the actual computer a different pattern of R,G and B is set up for logical colour number 1 in the palette memory. The command which performs this extremely useful trick is VDU19 which has five parameters associated with it and takes the form

```
VDU19,LC.AC,0,0,0
```

The first number after VDU19 is the logical number of the colour that we want to change and the second number is the actual colour that we want displayed on the screen. Thus to produce red text in mode 0 we might use the instructions.

```
100 MODE 0
110 VDU19,1,2,0,0,0
120 COLOUR 1
130 PRINT"This is red text in mode 0"
140 END
```

Here the VDU9 command has changed logical colour 1 (normally white) so that it now displays actual colour 2 which is red. We can in fact set more than one logical colour to the same actual colour. In mode 2 for instance we could easily replace the flashing colours (numbers 8–15) by the steady colours or we could make them all produce blue on the screen. A program for foreground colour switching is given in Fig. 5.3.

The other three parameters for VDU19 are there for future expansion and with the present operating system and hardware they do nothing. These three parameters must be included in the command, otherwise an error will be flagged so they are all set at 0. To save a little typing effort it is possible to shorten the VDU19 command to the following form:

```
VDU19,1,4;0;
```

In this form the semicolon after a number indicates that it will take up two bytes of memory. Here one of the three 0 terms is combined with the actual colour number (4 in this case) whilst the remaining two 0 terms are combined as the 0 followed by a semicolon. It is important here to include the final semicolon otherwise an error will be reported.

```
100 REM Foreground colour switching
110 REM using VDU19
120 MODE1
130 COLOUR1
140 GCOL0,2
150 PRINT TAB(3,3)"MODE1 "
160 PRINT TAB(3)"Colour switching"
170 PRINT TAB(3)"Text colour = 1"
180 PRINT TAB(3)"Graphics colour = 2"
190 MOVE 100,100
200 PLOT1,200,0
210 PLOT81,-200,200
220 PLOT81,200,0
230 FOR N=1 TO 64
240 CT=N+1
250 CG=N+2
260 VDU19,1,CT,0,0,0
270 VDU19,2,CG,0,0,0
280 FOR T=1 TO 5000:NEXT T
290 NEXT N
300 END
```

Fig. 5.3. A program for foreground colour switching.

```
100 REM Background colour switching
110 MODE1
120 GCOL0,129
130 CLG
140 VDU28,1,15,38,1
150 COLOUR128
160 CLS
170 COLOUR 2
180 PRINT TAB(0,0)"MODE 1"
190 PRINT"Background colour switching"
200 PRINT"Text background=128"
210 PRINT"Graphics background=129"
220 PRINT"Text colour=2"
230 PRINT"Graphics colour=3"
240 MOVE 100,100
250 PLOT1,200,0
260 PLOT81,-200,200
270 PLOT81,200,0
280 FOR N=1 TO 50
290 TB=N
300 GB=N+1
310 VDU19,0,TB,0,0,0
320 VDU19,1,GB,0,0,0
330 FOR T=1 TO 5000:NEXT T
340 NEXT N
360 END
```

Fig. 5.4. A program for background colour switching.

At this point we must consider what happens to background colours when we change the meaning of the logical colour numbers. In fact when we change the colour displayed by a foreground logical number the corresponding background colour will also change. Thus if we change the colour displayed by logical colour 0 then the background colour will change immediately. This assumes that the logical background colour has not been changed from its normal value of 0 by using a GCOL command. If the background had been set to colour 1 then changing the actual colour for colour 1 will cause the background colour to change. A program demonstrating background colour switching is shown in Fig. 5.4.

The advantage of being able to switch colours in this way is that the change is instant and affects everything that is drawn on the screen in a particular logical colour. When we set background colour you will remember that we used a COLOUR or GCOL command followed by clearing the screen to produce the new background colour. This unfortunately erases anything that is already drawn on the screen. If we use a VDU19 command to change the palette then we can switch the background from black to perhaps green instantaneously without affecting anything that is already drawn on the screen. Of course anything that has been drawn in green using the green foreground colour would disappear against a green background but it has a different logical colour number from the background. These foreground objects may be rendered visible again by changing their actual colour to something other than green by using VDU19.

Suppose we are in a four-colour mode but we wish to have blue as colour 1 instead of red. This can be achieved by using the VDU19 command as follows:

```
1100 VDU19, 1, 4, 0, 0, 0
```

and this will cause logical colour 1 to be blue in future in the program. Thus in a two-colour mode we can have any two colours as the foreground and background colours but only two colours can be produced on the screen at any time. In a four-colour mode we can select any four colours for display in place of the normal default colours black, red, yellow and white. Note that the background will always take up one of the logical colours in any mode.

Some interesting effects occur when the background colour is changed in mode 3 or mode 6. In these modes ten rows of dots are allocated to each row of text but only eight rows of dots are

controlled by the colour palette system. As a result if we use the command

```
VDU19,0,4,0,0,0
```

we will get a blue background with black lines across it, which makes the screen look like a piece of ruled notepaper. Of course we could use

```
VDU19,0,7,0,0,0
```

to produce a lined white screen and then use

```
VDU19,1,0,0,0,0
```

to make the text black so that the screen looks like writing paper. You could equally well choose to have yellow or pale blue green (cyan) paper and any contrasting colour for the text by using VDU19 commands.

More about GCOL

So far we have always used GCOLØ when setting up the graphics drawing colour and this as we have seen sets up the actual foreground colour that we are going to draw with. There are other variations of GCOL where the first parameter number is 1, 2, 3 or 4. These carry out various logical operations before putting the logical number into the display. These GCOL functions are as follows:

```
GCOL0,C    SC = C
GCOL1,C    SC = C OR EC
GCOL2,C    SC = C AND EC
GCOL3,C    SC = C XOR EC
GCOL4,C    SC = NOT C
```

where C is the new logical colour, SC is the logical colour produced on the screen, EC is the existing logical colour on the screen. Note that here we refer to the colours at the point or along the line being drawn. OR, AND, XOR and NOT are logical functions, whose actions on four-bit binary words are shown in fig. 5.5.

Each logical colour number consists of one, two or four binary data bits giving two, four or sixteen possible colours respectively according to the display mode selected.

GCOL4 is perhaps easiest to understand. The NOT function is also known as the *complement* or *invert* function and means that the state of each of the data bits in the logical colour number is changed

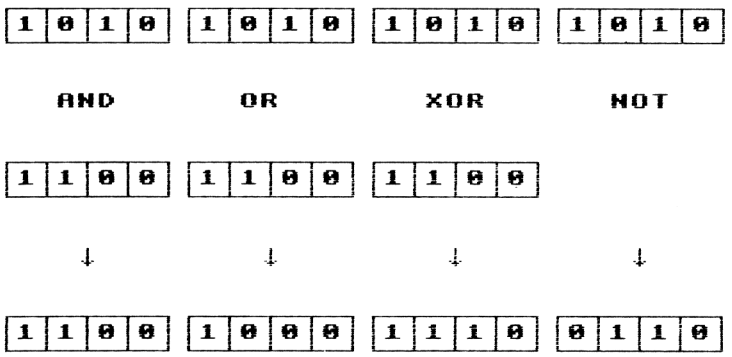


Fig. 5.5. Logic function actions on four-bit binary words.

from 0 to 1 or *vice versa*. Suppose we select colour 2 in mode 1 which will normally be yellow. The binary number is (1,Ø). If we use GCOL4 then this is complemented to give binary number (Ø,1) which produces red on the screen. In mode 2 the GCOL4 function would normally convert a steady colour into one of the flashing colours with numbers from 8 to 15.

If we use GCOL1 this causes the colour we specify to be ORed with the colour that is already on the screen at the drawing position. Now to understand the action of this command we need to look at the way in which the BBC Micro stores its information about the pixel colour. In mode 1 and other four colour modes two data bits in the memory word are used for each pixel and each word in memory represents four adjacent pixels on the screen. Each bit can be either on (1) or off (0). With two bits per pixel there are four possible combinations giving colour 0 (0,0), colour 1 (0,1), colour 2 (1,0) and colour 3 (1,1). Here the states of the two bits for the pixel are shown in brackets.

When we carry out a logical OR operation between two sets of two bits then if a bit in one word is a 1 the corresponding bit in the result will also be a 1 irrespective of the state of the same bit in the other word. Thus if the colour already on the screen is black (0,0) and we select GCOL,1 the bit in the red colour (0,1) that we are asking for overrides the 0 in the black word and the result will be a red colour (0,1) display. If however the colour already displayed were yellow (1,0) then a 1 bit will appear in the result from each of the colours to give (1,1) in the memory and the resultant display will be white. With an OR function our new specified colour effectively adds bits to the existing colour data to produce the new colour.

GCOL2 causes the new colour to be ANDed with the existing colour. Here there will be a 1 in the output colour word only when

the corresponding bit in both the existing and new colour words is at 1.

The AND function can effectively be used to switch off colours from the display. In mode 2 data bit 0 (the least significant) is red, bit 1 is green, bit 2 is blue and bit 3 selects the upper eight (flashing) colours. ANDing red with any existing colour that contains red will produce red only. If there is no red component in the existing colour the display will be black since all colour bits will be turned off. In the AND operation only those bits which are common to both logical colour words will be set at 1 in the new colour that is written to the screen.

Let us look at some examples of the AND operation. If we AND red (0001) with green (0010) there are no common bits so the result is black (0000). If we take white (0111) which contains red, green and blue and we AND it with green (0010) the result will be green (0010) since the red and blue bits are only at 1 in one of the two colour numbers.

GCOL3 produces the third basic logical function which is called EXCLUSIVE OR (or XOR in its shortened form). Here if the bits in the two words are in the same state (either both at 0 or both at 1) then the resultant bit from the output is at 0. If the bit states differ the result will be a 1. If we use a word with all bits at 1 as the GCOL3 colour then the existing colour will be complemented so that red would become cyan and so on. If the GCOL3 colour is 0 the display colour is unaffected. An interesting feature is that if the GCOL3 colour is the same as the one already being displayed then the result will be black or the background colour. This can be useful in animation since it will effectively erase the image already present.

Another useful fact is that VDU18 is identical in action to GCOL so that we could use either GCOL1,3 or VDU18,1,3 and they would perform the same function. Since GCOL has less characters, however, it would seem logical to use GCOL rather than VDU18 in practice.

Using the GCOL logical functions

One problem frequently encountered in games type programs is that we will want to have different objects passing in front of or behind one another on the screen. Let us consider the situation where we have a yellow planet and a red spaceship on a black background.

Now suppose we want to show the spaceship passing behind the planet.

If we draw the planet and spaceship using GCOL0 and we draw the planet first then the part of the spaceship overlapping the planet would be red and the spaceship would appear to be in front of the planet. If we drew the planet after the spaceship this would produce the desired effect because the planet colour will overwrite the red of the spaceship where the objects overlap. Now if we want to show the spaceship in a new position we will have to erase it and then redraw it in the new position. If we erased the spaceship by drawing it in black then in the area of overlap part of the planet would also be erased. The snag here is that we will have to erase and redraw both objects to get the proper result with the spaceship in a new position.

If we drew the planet using GCOL0 and the spaceship with GCOL1 then where they overlap the logical colour numbers are yellow (0011) and red (0001). When these are ORed together the result is logical colour (0011) which is yellow. As a result the spaceship is partly hidden by the planet although it has been drawn on top of the planet. Before moving the spaceship to a new position we need to erase it. If we use GCOL0,0 this would also erase part of the planet. To overcome this we might use the AND (GCOL2) with logical colour 2 (green). This erases the spaceship since there is no red component in colour 2. Unfortunately it also turns the overlap area green. To overcome this problem we would allocate the actual colour yellow to logical colour 2. Now the planet will be restored and the spaceship erased. We can now redraw the spaceship in red at its next position and provided that we use GCOL1 there is no need to redraw the planet. So by careful choice of logical colours and the use of GCOL1 we can have objects passing one another without interference. If you wanted a red planet and a yellow spaceship then you could use VDU19 to assign actual colour red to logical colour 3 and yellow to colour 1. Of course you would have to make logical colour 2 red as well. Fig. 5.6 gives a program listing which will demonstrate this application of GCOL1 and GCOL2.

The most useful of the other GCOL options is GCOL3 where the EXCLUSIVE OR function is carried out between the colour already present and the new colour. An interesting feature here is that if a line is drawn using GCOL3 and then the same line is redrawn with GCOL3 and the same colour it will be erased but will not erase those other coloured areas that it overlaps.

Considering the spaceship and planet picture when the spaceship is drawn using GCOL3, the first time the spaceship is drawn the part

```
100 REM Spaceship passing planet
110 REM using GCOL logic functions
120 MODE2
130 VDU19,2,3,0,0,0
140 X=640:Y=512
150 R=300
160 GCOL0,3
170 PROCCIRCL
180 FOR XS=50 TO 1100 STEP 50
190 YS=500
200 GCOL1,1
210 PROCSHIP
260 FOR T=1 TO 200:NEXT T
270 GCOL2,2
280 PROCSHIP
290 NEXT XS
300 GOTO 180
310 END
480 REM
490 REM Draw planet
500 DEFPROCCIRCL
510 MOVE X+R,Y
520 TH=2*PI/R
530 CT=COS(TH):ST=SIN(TH)
540 DX=R:DY=0
550 FOR N=1 TO R
560 X1=DX*CT-DY*ST
570 DY=DX*ST+DY*CT
580 DX=X1
590 DRAW X+DX,Y+DY
600 PLOT85,X,Y
610 DRAW X+DX,Y+DY
620 NEXT N
630 ENDPROC
640 REM
690 REM Draw spaceship
700 DEFPROCSHIP
710 MOVE XS,YS
720 PLOT1,100,0
730 PLOT81,0,50
740 PLOT1,-100,0
750 PLOT81,0,-50
760 ENDPROC
```

Fig. 5.6. Spaceship passing a planet using GCOL logic operations.

that overlaps the planet will be green. This is because both yellow and red contain the red colour bit and this bit will be set to 0 in the displayed colour. The green colour bit of the planet area is unaffected by the XOR operation so the result in the area of overlap will be green. The rest of the spaceship is drawn in red since the

background colour has both red and green bits set at 0. If the spaceship is now redrawn the red component is restored to the planet overlap area, thus returning the planet to its normal yellow colour. The spaceship is erased because we now have two identical colour codes which produce an all 0 result after the XOR operation. So by successively drawing objects using GCOL3 we can draw and erase them without destroying the image of other objects that they overlap. This is particularly useful in animation of objects.

Plotting in background and inverted colours

We have used PLOTØ and PLOT1 and also PLOT4 and PLOT5, but you may have wondered what happened to PLOT2 and PLOT3; and how about PLOT6 and PLOT7? Well there are such commands and they perform some interesting functions. These commands allow us to plot in different colours from the normal drawing or foreground colour.

If we use PLOT2 or PLOT6 then we draw the line in the complementary colour to the current foreground colour. Thus if we have selected logical colour 1 the PLOT2 command will actually draw in either colour 0, 2 or 14 depending upon the mode selected. This is equivalent to using GCOL4 before a normal drawing command.

PLOT3 and PLOT7 are rather more useful since they will cause the lines to be drawn in the current background colour (see Fig. 5.7). In effect this is an erase command because the line will now become invisible since it is the same colour as the background. Note that this is not always the same as drawing a line in colour 0 because the background may not be black. This is useful because we can now make the background any colour and erase lines using PLOT3 or PLOT7 without having to work out which colour the lines have to be in order to erase them. We can also erase an area of colour by using PLOT83 or PLOT87. Note that PLOT3 and PLOT83 use relative X,Y coordinates whilst the other two use absolute X,Y positions on the screen.

A single dot on the screen can be erased by using PLOT67 with relative X,Y or PLOT71 with absolute X,Y values.

```
100 REM Effect of PLOT82 and PLOT83
110 MODE1
120 GCOL0,129
130 CLG
140 COLOUR3
150 PRINT TAB(0,1)"Horizontal rectangle"
160 PRINT"in colour 3 with PLOT81"
170 MOVE 200,200
180 GCOL0,3
190 PLOT1,600,0
200 PLOT81,-600,200
210 PLOT81,600,0
220 PRINT:PRINT"Vertical rectangle"
230 PRINT"in colour 2 with PLOT81"
240 MOVE 400,100
250 GCOL0,2
260 PLOT1,0,600
270 PLOT81,200,-600
280 PLOT81,0,600
290 FOR T=1 TO 20000:NEXT T
300 VDU31,0,5
310 PRINT"in colour 2 with PLOT82"
320 MOVE 400,100
330 PLOT2,0,600
340 PLOT82,200,-600
350 PLOT82,0,600
360 FOR T=1 TO 20000:NEXT T
370 VDU31,0,5
380 PRINT"in colour 2 with PLOT83"
390 MOVE 400,100
400 PLOT3,0,600
410 PLOT83,200,-600
420 PLOT83,0,600
430 END
```

Fig. 5.7. Plotting in inverse or background colour.

Producing dotted lines

There are a whole series of PLOT commands which permit the drawing of dotted lines instead of the normal continuous lines. They have numbers running from 16 to 31 and are arranged in two groups of eight commands. The first group from 16 to 23 draw dotted lines from the cursor position to the position specified in the PLOT command. The second group from PLOT24 to PLOT31 also draw dotted lines but the last dot of the line is not plotted.

Of these commands the useful ones are:

PLOT17 and PLOT21 dotted line in selected colour (see Fig. 5.8).

PLOT19 and PLOT23 dotted line in background colour.

PLOT17 and PLOT19 use relative X,Y whilst PLOT21 and PLOT23 use absolute X,Y values.

```
100 REM Dotted and solid lines
110 MODE2
120 GCOL0,3
130 X=640
140 Y=200
150 R=600
160 DTH=PI/18
170 TH=0
180 FOR N=1 TO 9
190 PLOT4,X,Y
200 PLOT21,X+R*COS(TH),Y+R*SIN(TH)
210 TH=TH+DTH
220 PLOT4,X,Y
230 PLOT5,X+R*COS(TH),Y+R*SIN(TH)
240 TH=TH+DTH
250 NEXT N
260 END
```

Fig. 5.8. Drawing dotted lines using PLOT21.

To suppress the last dot of the line simply add 8 to the PLOT number. Thus PLOT29 draws a dotted line without the last dot using absolute X,Y coordinates.

A summary of all PLOT commands is given in Fig. 5.9.

Producing more colours

Normally we can draw in only eight basic colours if we ignore the flashing colours of mode 2. Whilst this number of colours is quite useful there will be times when it would be nice to have more colours available. This could be very useful if pictures are being produced for say an adventure type game. In fact it is possible to produce many more colours by simply mixing colours from the available set.

A simple approach to this would be to fill an area with colour by drawing a series of adjacent lines and having alternate lines in different colours. Let us try this with a simple figure such as a square. Instead of using the normal triangle colour fill we shall actually fill the square with lines by drawing them one above the other from the bottom of the square to the top. The program listed in Fig. 5.10 performs this task and uses two colours with which it draws

```
Simple cursor move commands

PLOT0
PLOT8
PLOT16
PLOT24
PLOT64
PLOT80

Drawing in selected colour

PLOT1  Solid line
PLOT9  Solid line without last point
PLOT17 Dotted line
PLOT25 Dotted line without last point
PLOT65 Single dot
PLOT81 Filled triangle to last 2 points

Drawing in inverse colour

PLOT2  Solid line
PLOT10 Solid line without last point
PLOT18 Dotted line
PLOT26 Dotted line without last point
PLOT66 Single dot
PLOT82 Filled triangle to last 2 points

Drawing in background colour

PLOT3  Solid line
PLOT11 Solid line without last point
PLOT19 Dotted line
PLOT27 Dotted line without last point
PLOT67 Single dot
PLOT83 Filled triangle to last 2 points

NOTE  These commands all use relative
coordinates. For PLOT operations where
the absolute X,Y coordinates are to be
used add 4 to the PLOT number (e.g.
PLOT1 is relative and PLOT5 is the
corresponding absolute coordinate
command).
```

Fig. 5.9. Summary of main PLOT commands.

alternate lines as it fills the square. After the square has been filled it is erased and then redrawn with a different pair of colours. The program sequences through all of the possible pair combinations of the eight basic colours to give 64 colours. Some of the colours

```
100 REM Colour mixing using lines
110 MODE2
120 CX=440:CY=320
130 W=400
140 GCOL0,7
150 MOVE CX-8,CY-4
160 PLOT1,W+16,0
170 PLOT1,0,W+8
180 PLOT1,-W-16,0
190 PLOT1,0,-W-8
200 FOR C=0 TO 7
210 FOR D=0 TO 7
220 FOR N=0 TO W STEP 8
230 MOVE CX,CY+N
240 GCOL 0,C
250 PLOT1,W,0
260 PLOT 0,0,4
270 GCOL0,D
280 PLOT1,-W,0
290 NEXT N
300 NEXT D
310 NEXT C
320 END
```

Fig. 5.10. Colour mixing using alternate lines.

produced are duplicates where the colours of the adjacent lines are simply interchanged and the resultant colour on the screen is the same for both combinations in such cases. This technique relies for its effect upon the fact that the eye tends to merge the colours of the adjacent lines to produce a new colour. On a normal television receiver there will probably be moving coloured patterns with some combinations of colour. This is due to the way the colour signals are processed in a television receiver and should not occur on a colour monitor.

Dotted lines may also be used to provide colour mixing. In this case a dotted line is drawn in one colour then a second dotted line is drawn over it but is shifted by one dot position along the X axis. The second line is drawn using GCOL1 to select the second colour. This effectively places dots of the second colour between the dots of the first colour. The result is similar to drawing solid adjacent lines which run vertically and a regular line pattern can still be seen.

To overcome the regular line pattern the adjacent dotted lines may be staggered by one dot position relative to one another so that the dots form a checkerboard pattern. This tends to give very good results even in mode 2 where the resolution is relatively low. In the program shown in Fig. 5.11 three colours are combined to produce

```
100 REM Colour mixing using dots
110 MODE2
120 CX=440:CY=320
130 W=400
140 FOR C=0 TO 7
150 FOR D=7 TO 0 STEP -1
160 FOR E=1 TO 7
170 FOR N=0 TO W STEP 8
180 MOVE CX,CY+N
190 GCOL 0,E
200 PLOT17,W,0
210 PLOT 0,8,4
220 GCOL1,E
230 PLOT17,-W,0
240 PLOT0,0,-4
250 GCOL1,D
260 PLOT17,W,0
270 PLOT0,-8,4
280 GCOL1,C
290 PLOT17,-W,0
300 NEXT N
310 CLG
320 NEXT E
330 NEXT D
340 NEXT C
350 END
```

Fig. 5.11. Listing to show production of more colours by mixing dots.

about 500 different colour combinations. Here one colour appears on alternate dots of all lines whilst the others fill in the blank spaces on alternate dotted lines.

Elastic band drawing program

By using GCOL3 we can produce a rather better sketching program than the one given in Chapter 1. You will probably have found with that one that even drawing a straight line is not so easy because of course we don't have a convenient ruler to run the pen along as we would if we were drawing on paper.

In the program listed in Fig. 5.12 we start off with a short line drawn near the centre of the screen. The end of the line at point X1,Y1 is fixed but the other end of the line (X2,Y2) can be moved around the screen by using the arrow keys. When the end of the line is moved the original line is erased and a new line drawn to the new position of X2,Y2. As the point X2,Y2 moves around the screen the

```
100 REM Elastic band sketching program.
110 *FX4,1
120 MODE1
130 CLG
140 X1=0:Y1=0
150 X2=X1+5:Y2=Y1+5
160 GCOL3,3
170 DRAW X2,Y2
180 A=GET
190 GCOL3,3
200 IF A=81 THEN 250
210 IF A=68 THEN GCOL0,3
220 IF A=69 THEN GCOL0,0
230 MOVE X1,Y1
240 DRAW X2,Y2
250 IF A=137 THEN X2=X2+5
260 IF A=136 THEN X2=X2-5
270 IF A=139 THEN Y2=Y2+5
280 IF A=138 THEN Y2=Y2-5
290 IF A=68 OR A=69 OR A=77 THEN X1=X2:
    Y1=Y2:X2=X2+5
300 GCOL3,3
310 MOVE X1,Y1
320 DRAW X2,Y2
330 GOTO 180
340 *FX4,0
350 END
```

Fig. 5.12. A drawing program using the elastic band technique.

line stretches and contracts in much the same way as if it were a piece of elastic.

When the line is placed where we want it then it can be fixed in place by hitting the D (draw) key to write the line permanently on the screen. Now the original free end of the line becomes the fixed point about which a new line can be drawn.

Sometimes of course we may just want to move to a new point on the screen without drawing the line. In this case the M (move) key is pressed to erase the line and place the fixed point at position X2,Y2 ready to draw a new line. With this scheme we have effectively produced a ruler for drawing straight lines.

The key to this program is the use of GCOL3. An initial line is drawn using GCOL3,3. One end is effectively fixed whilst the other end may be moved by using the arrow keys. Each time a key is pressed the line is redrawn using GCOL3,3, which effectively erases it. After the required function for the key has been carried out a new line is drawn.

The arrow keys merely cause the free end of the line to move

around. When the D key is pressed the current line is drawn in permanently. The E key causes the line to be erased permanently and the M key allows the drawing point to be moved to a new position. In fact the D, E and M keys all cause the fixed point of the elastic line to move to the old free point and we now start drawing a new line from where the last one finished.

The difference between M and E is that M does not affect any other lines already on the screen, whereas E erases all points along the elastic line. This is useful for erasing lines that have already been drawn.

Line 11Ø disables the arrow keys and allows them to return an ASCII code instead of moving the text cursor. Lines 14Ø–17Ø set up graphics coordinate values and draw an initial line on the screen near the centre. Line 18Ø reads the ASCII code number of any key that is pressed. Line 2ØØ detects a Q key and causes the program to quit. Lines 12Ø and 13Ø detect the D and E keys respectively and set the drawing colour to either white (D) or black (E). Lines 14Ø and 15Ø redraw the line. If neither D nor E has been pressed the line already drawn is erased because the GCOL3,3 in line 1ØØ is still effective.

Lines 25Ø–28Ø detect the cursor arrow keys and adjust the values of X2 and Y2 to move the free end of the line to a new position. Line 29Ø detects the D, E or M keys and if any of these were pressed then X1,Y1 are updated to be equal to X2,Y2. This moves the fixed end of the line to the new position. X2 is also increased to produce a new short line ready to be moved about. Lines 3ØØ, 31Ø and 32Ø draw the new line and then the program loops back to wait for another key input. Finally, before exiting the program restores the operation of the arrow keys by using *FX4,Ø.

Chapter Six

The Teletext Mode

Mode 7 on the BBC Micro is rather different in operation to the other display modes, particularly where colour and graphics are concerned. It is known as the teletext mode and is designed to be compatible with the teletext services such as Ceefax and Oracle which are transmitted along with the television programmes from the BBC and ITV. It is also compatible with viewdata services such as British Telecom's Prestel, where data is transmitted via the public telephone network.

In the teletext mode you will notice that there are some differences in the text character set. Some of the bracket symbols have changed and there are now some fractions which can be displayed. The reason for this is that in mode 7 the BBC Micro uses a different system for generating the teletext type display on the screen. Teletext is basically a text display mode although as we shall see, relatively low resolution graphics can be produced by using the mosaic graphics technique.

When the BBC Micro is working in modes 0 to 6 it actually stores in its display memory the dot pattern for each character being displayed. These dot patterns are actually held as a table in another part of memory but each time you use the machine to print some text on the screen it will call up the appropriate dot pattern and write it into the screen memory.

Now this approach to displaying text does allow very flexible mixing of text and graphics on the same screen, but the price paid for this is that a large part of the available computer memory is tied up for the display.

A very important difference when you select mode 7 is that the computer now uses a dedicated character generator chip. In fact it is the same chip that is used in many television receivers that have a teletext capability. The character generator will in fact produce the dot pattern as the television screen is scanned and needs only a single

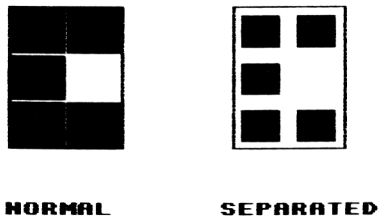


Fig. 6.1. Typical mosaic graphics symbols in mode 7.

8-bit data word for each symbol to tell it which pattern of dots to produce. Now instead of having to store all the dot patterns we need only have a single memory word for each displayed symbol. To produce the display on the screen the symbol code for each successive text symbol is read from the display memory and applied to the special teletext character generator chip. This chip then selects the required dot pattern and produces the appropriate video signal to produce the dot pattern on the screen. Typical mode 7 mosaic graphics symbols are shown in Fig. 6.1, and the bit allocation and value for mosaic graphics symbols in Fig. 6.2.

The screen format for teletext is 24 rows of 40 characters per row and since only one byte of data is needed to define a character the total memory requirement for the display is 960 bytes. In fact 1000 are actually allocated because the BBC Micro does allow 25 rows of text in this mode. This is a great saving, however, compared with the 1ØK or 2ØK of memory required for the other modes.

There are no high resolution graphics facilities in the teletext mode since it is basically a text display mode. We shall see in a moment, however, that graphics displays can be produced, albeit in relatively low resolution.

Bit 1 1	Bit 2 2
Bit 3 4	Bit 4 8
Bit 5 16	Bit 7 64

Fig. 6.2. Bit allocation and value for mosaic graphic symbols in mode 7.

Colour selection

As with the other display modes, when mode 7 is first selected the display will automatically be set up for white text on a black background. Unlike the other modes the teletext mode will not respond to commands such as COLOUR and GCOL. We can in fact produce colour displays in this mode but the technique for setting up colours is somewhat different.

Whereas the other display modes actually store a pattern of bits in the display memory to define which colour is to be produced, the teletext mode merely stores symbol codes so a different technique is used to set up colour. At the start of each row of text the colour is automatically set at white. If we want to change the text colour a control code is inserted into the row in one of the text symbol positions. This control code tells the display chip that the colour of the following text is to be changed. The display system is so designed that these extra control characters are displayed as blank spaces on the screen. One disadvantage of this control code approach is that each time the text colour is changed in a row one character space is lost, which reduces the amount of text that can be displayed across the row. Usually this lost space will be at the left-hand edge of the display.

The control codes for text colours are:

128 Black text
129 Red text
130 Green text
131 Yellow text
132 Blue text
133 Magenta text
134 Cyan text
135 White text

The control codes are placed into the display by using a PRINT statement in the same way as for printing text, but because they are special codes the CHR$ form has to be used. Let us see how this works by entering some direct commands.

First select the teletext mode by typing

```
MODE 7
```

If the cursor is off the top of the screen then type

```
*TV 254
MODE 7
```

which should move the cursor down on to the screen. Next type in

```
PRINT CHR$(129);"RED TEXT"
PRINT CHR$(130);"GREEN TEXT"
```

At this point the two messages should have appeared with the first in red and the second in green.

If we want to change the colour in a single line of text a new colour control code is inserted before the second segment of text, as follows:

```
PRINT CHR$(129);"RED TEXT";CHR$(134);"CYAN TEXT"
```

This will produce red and cyan coloured text on the same row. The text colour will apply only to text following the control code in one row. At the start of the next display row the text colour is automatically reset to its normal white colour. This means that if every row is to have coloured text then each row must start with a colour control code.

Now typing in all those CHR$ commands is rather tedious. Fortunately on the BBC Micro there is another way to send text and control codes to the screen. This uses the VDU command followed by the numbers representing the codes. So we could equally have used

```
VDU 129:PRINT"RED TEXT";:VDU 134:PRINT"CYAN TEXT"
```

The TAB command works in the same way for the teletext mode as it does in other modes so that we can write text to any part of the screen by using TAB(X,Y). The thing to remember here is that if you want coloured text then you must allow a space before the text symbol for the colour control code. Try running the program listed in Fig. 6.3. This will write 100 random text symbols in different colours all over the screen by using random X,Y values in the TAB (X,Y) term of the PRINT statement. The maximum value for X is 39 and for Y it is 24.

Setting background colour

In the other modes we were able to set up the background colour by using either COLOUR or GCOL followed by clearing the screen. The background colour for the teletext mode is normally set at black as in the other modes. We can in fact set up the teletext background colour to any of the available eight colours, but the process is rather

```
100 REM Random coloured letters
110 MODE 7
120 FOR N= 1 TO 100
130 X=RND(39)-1
140 Y=RND(24)-1
150 C=128+RND(7)
160 TS=32+RND(50)
170 VDU31,X,Y
180 VDU C,TS
190 NEXT N
200 END
```

Fig. 6.3. Program to produce random coloured letters in mode 7.

more complicated than for the other modes.

When teletext was first devised the background colour could only be black. When it was decided that different coloured backgrounds would be a nice idea the problem was that most of the 32 available control codes had already been allocated for other purposes and only two could be made available for background colour control. This problem was overcome by using one control code to set the background colour to the current display colour. The second control code was used to switch back to the normal black background. As with other control codes the background will only be affected during one row of the display. If the whole screen background is to be changed then control codes are needed at the start of every row.

The two control codes to deal with background colour are

```
157  SET NEW BACKGROUND
156  BLACK BACKGROUND
```

At the start of every display row the background is automatically set to black and the text to white. If the first control code at the start of a row is 157 then the background will change to white. If a different colour background is required then the text must first be set to the required background colour and then the 157 code is used to set the background colour. Try typing in the following, after selecting mode 7:

```
VDU 157:PRINT"WHITE TEXT"
```

You will notice that the whole line, including the space containing the control code, now has a white background but what has happened to the text message?

When we insert a new background control code the background will change to the current text or graphics display colour. Any

following text will be in the same colour so it will be invisible. To deal with this we must have a colour change control code after the background set control to make the text a different colour. Try the following:

```
VDU 157,129:PRINT"RED ON WHITE"
```

This should produce red text on a white background. If we had wanted a yellow background then the text colour would first have to be set at yellow as in the following command:

```
VDU 131,157,129:PRINT"RED ON YELLOW"
```

which should produce red text on a yellow background. As you can see, this process has already used up three of our available forty characters in the text row. This may seem rather complicated but is unfortunately a consequence of the limited number of control codes available in the teletext mode.

The black background code merely returns the background to its normal black colour. This can be seen in the following command:

```
VDU 131,157,129:PRINT"RED ":VDU 156:PRINT
```

Here the background switches from yellow to black after the text message.

A program illustrating the use of background colour in mode 7 is given in Fig. 6.4.

```
100 REM Starry night
110 MODE 7
120 FOR N=1 TO 24
130 PRINT CHR$(132);CHR$(157)
140 NEXT N
150 FOR N=1 TO 20
160 X=RND(36)+1
170 Y=RND(20)-1
180 VDU31,X,Y
190 PRINT CHR$(135);"*";
200 NEXT N
210 VDU31,10,24
220 PRINT"Starry night      ";
230 END
```

Fig. 6.4. Use of background colour in mode 7.

Graphics in mode 7

Although the teletext mode is primarily a text display mode it is possible to produce graphics pictures. In the teletext mode graphics displays are produced by using the mosaic graphics technique. An extra 64-character symbol set is provided for this purpose. These extra symbols are produced by dividing up the symbol space into six smaller blocks arranged as two columns of three blocks each, as shown in Fig. 6.1.

With 40 characters per row and two blocks across each graphics symbol this gives 80 blocks across the screen. The 25 text rows with three blocks in each symbol gives 75 blocks down the screen. Since a control code is needed to select graphics at the start of a text row the maximum number of graphics elements across the screen is reduced to 78, giving an overall graphics resolution of 78×75. This is much less than we have in the high resolution modes but nevertheless quite respectable pictures can be produced by using these rather coarse graphics displays. This can be seen by looking at some of the graphics displays produced on the broadcast teletext services. Although there are 25 display rows it is generally advisable to use only 24. If you use the 25th row and end with a return the screen will scroll so that all of the text moves up one line to leave an empty line free at the bottom for new input. If the display scrolls you will lose the top line of the original picture. Broadcast teletext uses only 24 lines and it is as well to stick to that format.

Selecting the graphics mode and the graphics display colour is achieved in much the same way as we selected a text colour. A control code is inserted into the display row to switch from text to graphics. This technique is used because there are not enough spare character codes to cope with the set of graphics symbols. In fact the graphics symbols share the same character codes as the text symbols. When the graphics mode is selected the following symbol codes select mosaic block patterns instead of text symbols.

There are in fact seven different control codes to select graphics. Each of these codes selects not only the graphics symbol set but also the display colour so they operate in a similar fashion to the text colour control codes. The codes for selecting graphics have numerical values from 145 to 151 and have the following functions:

145 Red graphics
146 Green graphics
147 Yellow graphics

148 Blue graphics
149 Magenta graphics
15Ø Cyan graphics
151 White graphics

The change from graphics back to text is achieved by simply inserting a text colour control code which switches the display mode back to text. Mixing graphics and text along the same line is quite straightforward but for each change one symbol space is lost because it is occupied by the control code.

Let us see how this works. Try typing in the following command:

```
vdu 147:PRINT "Graphics";CHR$(134);"TEXT"
```

You should have a yellow G, then a pattern of yellow blocks followed by cyan text. The word 'graphics' has in fact been displayed as the equivalent mosaic graphics symbols.

The mosaic symbols

Each of the six blocks that make up a graphics symbol is allocated one data bit in the 8-bit character code. These are bits 1–5 and bit 7. Bit 6 is not used in this case and there is a very good reason for this. In the example we tried a moment ago you will notice that although the lower case letters of the word 'Graphics' were replaced by graphics patterns the capital G appeared normally. This is achieved by using bit 6 to allow capital letter text to be displayed amongst graphics symbols without the need to switch back to the text mode. This feature can be very useful at times, but it does make the allocation of bits to the mosaic blocks rather illogical.

Since each block corresponds to one bit then each has a particular numeric value when it is lit and 0 when it is unlit. We can therefore arrive at the actual graphics code for any mosaic symbol merely by adding together the values of those blocks that are lit in the pattern we want to display. This can be useful if you want to use the computer to calculate which code numbers it requires to produce a particular mosaic pattern. For most purposes it is easier to look up a table which shows the pattern of the symbol and the corresponding code number. There is a very useful table of this kind in the *User's Manual* for the BBC Micro.

Setting up graphics symbol data

One disadvantage of using the CHR$ type of notation for feeding in the graphics character codes is that it involves a lot of typing. Also, since the lines are longer the BASIC interpreter takes more time to execute the line. This means it will take longer to draw the picture on the screen.

There are two alternative approaches to this situation. One is to convert the graphics into a string first by using an instruction of the form

```
100 A$=CHR$(  )+CHR$(  )+CHR$(  )+ etc.
```

and then in the PRINT line we simply have

```
110 PRINT A$
```

or alternatively we can use the VDU command. If we use VDU followed by a string of numbers this is equivalent to PRINT followed by a series of CHR$ terms. So now our graphics line might become:

```
100 VDU 147,190,210 etc
```

The only disadvantage of this method compared with creating a string A$ is that the string of graphics data is not retained in memory. Thus the A$ string approach is best if the same pattern of graphics codes is likely to be required on several lines.

When there is a lot of data the best approach is to set up a DATA list and then use a READ statement in a loop to read the data and set up the character string as follows:

```
100 A$=""
110 FOR N = 1 TO NDATA
120 READ A
130 A$ = A$ + CHR$(A)
140 NEXT N
150 PRINT A$
```

Note here that a numeric variable A is used for reading the data and this is converted to a string by CHR$ and added to the string A$. This is probably the best approach for reading in the data to produce a full screen graphics picture in this mode. Note that you could also read the data from a cassette file if desired.

A different kind of graphics

There are in fact two versions of the graphics symbol set. The normal set of symbols has the six blocks inside the character space touching one another and is called the *contiguous* graphics symbol set. The alternative graphics symbol set also has six mosaic blocks but here a small border is left around each graphics block element. This is called the *separated* graphics set. Two control codes are used to select the particular type of graphics symbol to be displayed and these are:

```
153  Contiguous graphics symbols
154  Separated graphics symbols
```

At the start of a display row the contiguous symbols are set up so that if graphics are selected this is the type that would be displayed. To see the difference between the two sets try the following little program:

```
100 MODE 7
110 VDU31,0,11
120 VDU 150
130 FOR N=160 TO 180
140 VDU N
150 NEXT N
160 PRINT:PRINT
170 VDU 150,154
180 FOR N=160 TO 180
190 VDU N
200 NEXT N
210 PRINT
220 END
```

These separated graphics symbols can be useful in building up a picture because they have a different texture from the normal graphics symbols.

Producing flashing symbols

In mode 2 we can readily produce flashing symbols by choosing a colour in the range 8–15 and in other modes we can use the VDU19 command to set up a flashing text colour. Teletext has no COLOUR or VDU commands for this purpose but it is possible to have flashing text or graphics symbols.

At the start of each row of the display the text is always displayed

normally. To make a section of text or graphics in a row start flashing on and off we simply insert a control code in the symbol space at the point where the symbols are to be flashing. If we just want one word in a row to flash then after the word another control code is inserted which returns the following symbols in the row to the steady display state again. The control codes for this are

```
136 for flashing text or graphics
137 for steady text or graphics
```

Note that in teletext mode the symbols simply flash on and off, whereas in mode 2 the flashing is between the selected colour and its complementary colour. The flashing symbols are useful for drawing attention to a particular item but should not be overdone since a flashing display tends to be rather uncomfortable to watch.

Double height text and graphics

For some types of display it is useful to be able to have some text in larger letters than the rest of the display. The teletext mode allows the display of double height text or graphics symbols as well as the normal size symbols. At the start of each row single height is always set up. As with most options in the teletext mode there are two control codes which allow selection of either double or single height text or graphics symbols. These are

```
140  Normal height symbols
141  Double height symbols
```

An example of this can be seen by typing in:

```
100 VDU 130,141:PRINT"DOUBLE";:VDU140:PRINT"SINGLE"
110 VDU 130,141:PRINT"DOUBLE";:VDU 140:PRINT"SINGLE
```

Note here that you have to repeat the data on two successive lines of the display to get the correct results.

It is possible to mix double and single height in a line. To return to single height after a double height word the single height command must be inserted.

A point to note is that if single height is used with double height text the following row cannot be used for single height text.

Even bigger text

Apart from double height text it is often desirable to have even larger symbols for the page title. Many examples of this will be seen on the broadcast teletext services where almost every page uses a banner title of very large symbols. These large letters are simply built up from the graphics symbol blocks. An example of this is shown in the program listing of Fig. 6.5.

```
100 REM BBC Logo in Mode 7
110 MODE 7
120 A$="":B$="":C$=""
130 FOR N=1 TO 15
140 READ A,B,C
150 A$=A$+CHR$(A)
160 B$=B$+CHR$(B)
170 C$=C$+CHR$(C)
180 NEXT N
190 VDU31,0,11:VDU 147,157:PRINT
200 PRINT A$
210 PRINT B$
220 PRINT C$
230 VDU 147,157:PRINT
240 DATA 147,147,147,157,157,157
250 DATA 148,148,148,255,255,255
260 DATA 175,172,252,253,247,191
270 DATA 160,160,160,255,255,255
280 DATA 175,172,252,253,247,191
290 DATA 160,160,160,232,234,170
300 DATA 191,181,253,239,160,254
310 DATA 180,160,165
320 END
```

Fig. 6.5. Program to produce banner size text in mode 7.

Graphics hold

Because the colour and other control codes are embedded in the displayed text and graphics one problem does occur when we want to make a change in the colour of a graphics area. A control code must be inserted to change the colour and this will normally be displayed as a space which will leave an unsightly gap in the graphics display. This can be overcome by using a feature known as *graphics hold.*

When graphics hold is selected then if a control code occurs in the

middle of some graphics symbols the space which it occupies is filled by repeating the last graphics symbol to be printed before the control code. The control code for graphics hold is normally inserted close to the start of a text row.

The two control codes used to select and disable graphics hold are

```
158 Hold graphics
159 Release graphics
```

If the hold graphics mode is selected by using CHR$ (158) early in the row then when graphics are selected and a colour change occurs the space normally occupied by the control code will be filled by repeating the last used graphics code in that row. This will in fact be displayed in the old colour. The effect of graphics hold is shown by the listing of Fig. 6.6.

```
100 REM Graphics hold demonstration
110 MODE7
120 A$=""
130 FOR K=1 TO 4
140 A$=A$+CHR$(255)
150 NEXT K
160 FOR N=1 TO 23
170 PRINT TAB(0,N-1) CHR$(158)
180 NEXT N
190 FOR N=1 TO 23
200 VDU31,1,N-1
210 FOR K=1 TO 7
220 C=K+144
230 PRINT CHR$(C);A$;
240 NEXT K
250 NEXT N
260 VDU31,5,23
270 PRINT"WITH GRAPHICS HOLD"
280 FOR T=1 TO 10000:NEXT T
290 CLS
300 FOR N=1 TO 23
310 VDU31,1,N-1
320 FOR K=1 TO 7
330 C=K+144
340 PRINT CHR$(C);A$;
350 NEXT K
360 NEXT N
370 VDU31,5,23
380 PRINT"NORMAL GRAPHICS"
390 FOR T=1 TO 10000:NEXT T
400 GOTO 160
```

Fig. 6.6. Demonstration of the effect of graphics hold.

In the hold graphics mode, text can also be inserted into the graphics area without losing spaces for the control codes. Some care is needed in using this feature, however, and generally it is arranged that the graphics code which is being repeated is a solid block.

Drawing on the teletext screen

Although the teletext mode can produce graphics it does not have the equivalent of DRAW, PLOT and MOVE commands and is not particularly suited to drawing graphs or charts in the same way as the other modes are. It is possible to calculate where each graphics block element is on the screen and by choosing the appropriate symbol for the text position it is possible to light individual points on the screen and plot a graph. This is however a rather complex business and there seems little point in the exercise when perfectly good graph and chart drawing facilities are available in the other modes.

```
100 REM Random coloured blocks
110 MODE 7
120 FOR N=1 TO 24
130 PRINT CHR$(158)
140 NEXT N
150 FOR N=1 TO 5000
160 X=RND(37)
170 Y=RND(24)-1
180 VDU31,X,Y
190 C=RND(7)+144
200 PRINT CHR$(C);CHR$(255);
210 NEXT N
220 END
```

Fig. 6.7. Program to produce random coloured blocks in mode 7.

Teletext is quite useful for providing pictures of the locations in an adventure type game since the pictures take up little memory space. Here the pictures can be constructed by marking off a sheet of graph paper with the screen pattern of symbol spaces and individual graphics block elements are then filled in to produce the picture. This can then be translated into a data list and read in as required using a READ loop and building strings of characters for printing on successive lines of the display to produce the picture (see Fig. 6.7).

```
100 REM Random coloured dots
110 MODE 7
120 FOR N=1 TO 24
130 PRINT CHR$(158);CHR$(154)
140 NEXT N
150 FOR N=1 TO 5000
160 X=RND(36)+1
170 Y=RND(24)-1
180 VDU31,X,Y
190 C=RND(7)+144
200 S=RND(96)+159
210 S=S OR 32
220 PRINT CHR$(C);CHR$(S);
230 NEXT N
240 END
```

Fig. 6.8. Program to produce coloured patterns using segmented symbols in mode 7.

The possibilities for creating quite attractive pictures in the teletext mode are great and a look through the broadcast teletext magazines will give some idea of what can be achieved. Fig. 6.8 provides a further example.

Chapter Seven
Making Things Move

For many applications and in particular for games, we will want to produce moving objects on the display. To achieve movement or animation the object is simply displayed in a slightly different position every $1/10$ second or so. Because the eye cannot follow fast changes the image appears to move smoothly across the screen. This is the basic principle used in producing cartoon films.

In Chapter 5 where we had an elastic line which could be moved around the screen, we had effectively used the basic technique of animation on a computer graphics display. In games type programs various objects, such as spaceships, rockets or alien creatures may be moved around the screen by successively erasing and then redrawing the object in a new position further along the line of travel. This was demonstrated in Chapter 5 in the example of the spaceship passing behind the planet. Sometimes, however, the drawing is changed according to the direction in which the object is moving. For instance, if we have an aeroplane on the screen its image will need to be redrawn if the plane changes direction so that the nose is pointed in the direction of motion.

For more realistic results the object on the screen may need to change shape as it moves. An example of this would be a man walking across the screen. If the image of the man were left constant he would appear to glide across the screen. To give the impression of walking or running the position of the legs in the drawing of the man must be changed as the image moves from one position to another on the screen. For a bird flying on the screen the wings will need to flap and for realism this is not just a simple up and down motion but the whole shape of the wing changes as it makes a beat in the air.

A simple moving ball

For many games type programs we will want to move a simple object

such as a ball around the screen and this is conveniently done by using a text symbol for the ball. We could perhaps use an 'O' but in the following examples we shall use an asterisk (*).

We saw in Chapter 1 that by using the TAB(X,Y) function in a PRINT statement we could place a symbol in any desired character space on the screen. To produce animation the symbol is moved successively from one character space to the next either horizontally, vertically or diagonally, at high speed.

Horizontal motion is quite straightforward since all we have to do is add one to the X value to move right or subtract one from X to move left. The symbol may then be printed again. A point to note here is that the actual cursor will have moved one position to the right after printing the symbol. As the object moves around the screen we shall have to keep track of its position. This can be done by storing the current X,Y position of the moving object. These stored position values may be used in conjunction with TAB and PRINT to place the object in its new position and erase the last image. For

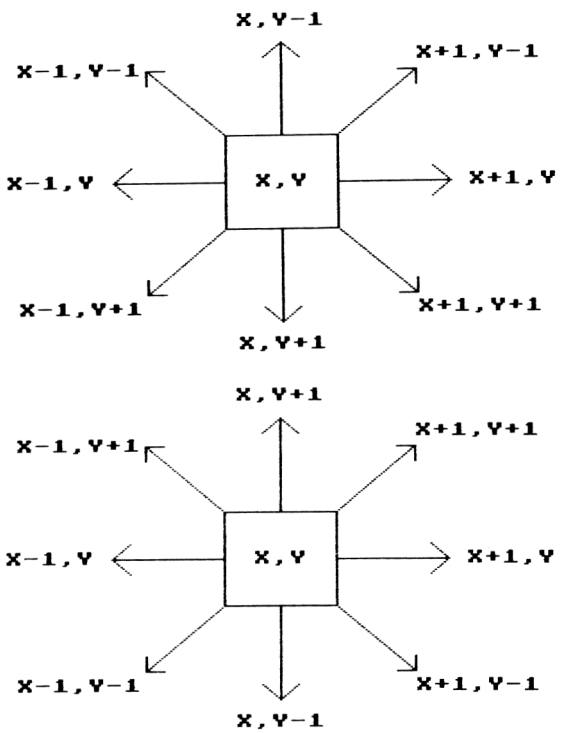


Fig. 7.1. (a) X,Y coordinate changes for motion in various directions using TAB(X,Y). (b) X,Y coordinate changes for motion when using PLOT, MOVE or DRAW.

vertical motion we add one to Y to move down and subtract one from Y to move up.

Diagonal motion is a little more difficult. Here we have to alter both X and Y values using a combination of either add or subtract one. Thus to go up to the right we need X+1,Y−1. Up to the left is X−1,Y−1 and for the downward motions we will need Y+1 with either X−1 for moving left or X+1 for moving right. Fig. 7.1 shows the arrangement for all eight directions. In these horizontal, vertical or diagonal moves the values of X and/or Y are altered one at a time.

It would also be possible to produce some intermediate directions by altering one value by two and the other by one, for example X+2,Y−1 give a direction just above horizontal to the right. A problem in using this type of motion is that the distance travelled varies according to direction.

Having moved the symbol we need to blank out the previous position and this is easily done by printing a space at position X,Y. Finally, we need to update the current position X,Y to point to the new position of the symbol ready for the next animation step. So the complete sequence for moving our * one position diagonally upwards to the right is

```
100 PRINT TAB(x+1,y-1)"*";
110 PRINT TAB(X,Y)" ";
120 X=X+1:Y=Y-1
```

A more convenient scheme is to make the change in X a new variable DX and the change in Y becomes DY. Now these can be set to −1, 0 or +1 as required and the printing operation can become a procedure as follows:

```
500 DEFPROCMOVE
510 PRINT TAB(X+DX,Y+DY) "*";
520 PRINT TAB(X,Y) " ";
530 X=X+DX:Y=Y+DY
540 ENDPROC
```

Bouncing off the walls

If you tried moving the ball with the simple program above and started from the centre of the screen all would be well until the ball reaches the edge of the screen. Here the message 'Illegal quantity' will occur as the program crashes. The reason is that X or Y has gone beyond its permissible limit value. To avoid this state of affairs we

need to arrange that when the ball reaches an edge of the screen it is reflected back towards the middle as if it had bounced off a wall.

This is quite easy to do by comparing X+DX and Y+DY, the new position for the ball, with the limits for X and Y. Suppose we are using mode 1 where X is 0 to 39 and Y is 0 to 31. First we check for X+DX = 0 and then for X+DX = 39 and if either of these is true we change the sign of DX. Similar checks are made on Y+DY using the values 0 and 31 and again if either is true the sign of DY is changed. The test routine becomes:

```
400 IF X+DX <> 0 AND IF X+DX <> 39
 THEN 420
410 DX=-DX
420 IF Y+DY <> 0 AND IF Y+DY <> 31
 THEN 440
430 DY=-DY
```

Now the * will turn back without actually entering the final character position at the edge of the screen. This is useful if you have drawn a border around the play area since it prevents the border being overwritten by * symbols and then being erased.

Normally the text cursor will move to the position after the printed symbol and erase anything that is already there but this is easily overcome by suppressing the text cursor at the start of the program by using

```
10 VDU23,1,0;0;0;0;
```

When the * is travelling horizontally or vertically it is reflected straight back off the boundary, but when motion is diagonal it comes back at an angle to its original movement. Thus if the ball is travelling up to the right and hits the right wall it will now travel left but continue to move upward. At a corner the * bounces off two walls and comes back on a course parallel to its original one.

Reflection off a bat

In games such as *Breakout* the ball bounces off a movable bat and depending upon where it hits the bat will travel straight or diagonally after reflection. Here we need to keep track of the bat position by using say BX and BY. In our tests for reflection off a wall routine we now include a check to see if X+DX,Y+DY is equal to BX,BY. If they are equal the reversal of direction is arranged by reversing DY. We might also check for BX+1,BY and BX−1,BY,

then produce diagonal motions by setting up the required values for DX and DY. In a game such as *Breakout* the bottom wall, containing the bat, may be checked for coincidences with the ball and if there is a match the ball is lost and the game ends or a new ball is used up. Here again the check is simply made for concidence between Y+DY and 31 and if this is true a further check is made against BY to see if the bat is present.

Collisions with other objects

In many games type programs, such as *Space Invaders*, the object is to fire missiles or drop bombs on other objects to destroy them. This means that we must detect when the missile reaches the same position as a target object. One technique for this is to maintain a table of the X,Y positions of the objects on the screen. Before moving the missile to its new position a check is made by comparing this position with that of each of the other objects in turn. If a match occurs, the program will branch to a subroutine or procedure which produces the required explosion effect and erases both the missile and the target object. Some housekeeping operations will now be required to delete the target from the list and perhaps shuffle the list to make the next search easier by moving up the X,Y values for other targets in the table. The process of keeping track of targets and missiles using this technique can become complicated.

A technique often used on computers for target–missile coincidence checks is to read out the character code for the symbol occupying the space where the missile is about to go. This can often be done by simply peeking the video memory, but is not practicable with the BBC Micro in modes 0 to 6 because of the way in which symbol data is stored in the memory. The BBC Micro actually stores the pixel pattern for the symbol rather than an ASCII code. It might be possible to compare the pixel pattern with the known pattern for the object being checked, but this is difficult. By using the machine operating system it is possible to check which particular symbol is present at the position of the text cursor, but again this seems to be a relatively complex process.

Making new symbols

In most cases we shall want something better than one of the

available set of text symbols for our moving object. We might for instance want to have one of the familiar invaders or 'puckman' type figures. In the BBC Micro there is a facility for creating new text symbols to your own custom requirements. We can make use of these new symbols to make up the moving object.

Suppose we want to have a small moving man. The man can be created as a new text symbol which will have a code in the range 224–255 whereas the normal text codes fall in the range 32–127. To produce the image on the screen we simply use the command

```
PRINT CHR$ (N)
```

where N is the code number we have allocated to the new symbol.

How do we create one of these new symbols? This may seem a relatively complicated business so let us go through it step by step.

The symbol space is effectively divided up into eight squares horizontally, and there are eight rows of dots in the symbol. The first step is to draw up the image you want by filling in the required dots in an 8 × 8 grid as shown in Fig. 7.2.

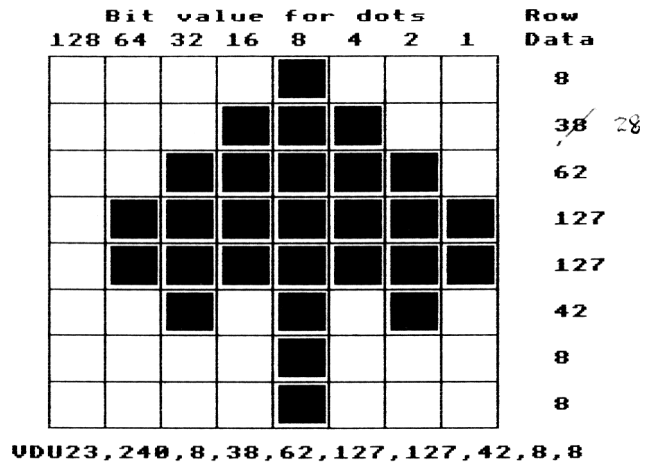


Fig. 7.2. Layout of dot pattern for a special symbol and the bit and word values related to the pattern.

In the BBC Micro the symbol dot pattern for these custom symbols is stored in memory as a set of eight data words. Each word represents one row of dots in the character pattern. In each word those dots in the row which are to be lit are set at 1 and the dots that are dark are represented by bits set at 0. Each data bit in the word has a binary numeric value. This starts with the bit at the right-hand end having a value of 1. The next bit to the left has a value of 2 and the

rest double in value for each bit, so that the left-hand bit is equivalent to 128.

To calculate the numerical value for each of the data words representing the symbol we simply have to add together the values of all of the dots in the row that are lit. This has to be done for each row of dots starting at the top row and working down. At the end of this stage we should have eight numbers. This is shown in Fig. 7.2, where the dot pattern for the symbol and the numeric values of the data words are shown. Next the data pattern for the new symbol must be set up in the computer memory.

To set up the pattern in memory we use the VDU command

```
VDU 23
```

This has the form

```
VDU 23, code,n1,n2,n3,n4,n5,n6,n7,n8
```

where CODE is the code value you want to use for the symbol and this must be in the range 224 –255. The set of eight numbers after the CODE are the numerical values for the eight rows of dots in the new symbol.

Suppose the pattern we want to produce is too complex to be properly shown as a single character. The solution here is to make up the desired symbol by using two special symbols side by side. In this case draw up two character grids alongside one another and mark on them the pattern of dots. Then the left half of the pattern might be defined as character code 225 and the right-hand half as character code 226. To produce the object on the screen we simply use the command

```
PRINT CHR$(225);chr$(226)
```

whenever we wish to display the symbol.

There are some problems with using multiple symbols to produce an object. When the object is being moved around the screen using the TAB commands it is important to check that the print position is not at the right edge of the screen, otherwise part of the object will automatically be shifted down to the start of the next line. Therefore a check must be made of the character position before trying to print the symbol and if it is at the edge some action needs to be taken to correct for the possible error in the display. If you wish to have true wrap around where the object reappears on the same line but at the other side of the screen then you will need a different PRINT routine for this condition and you would print the two parts of the symbol

individually at the points where you wish them actually to appear.

Sometimes the complete object may be built up from two symbols printed one above the other and moved as a pair. It is also possible to produce more complex objects using a group of four special symbols. Whilst these more complex patterns are fairly easily set up they can present problems if you are going to try to produce fast animation since there will now be two or four symbols to deal with.

Fig. 7.3 shows a simple program to produce an animated moving

```
100 REM Animated man
110 MODE1
120 VDU23,1,0;0;0;0;
130 VDU23,240,0,56,56,16,16,24,52,50
140 VDU23,241,24,16,24,24,24,16,48,48
150 VDU23,242,0,56,56,16,16,56,52,50
160 VDU23,243,24,20,28,24,48,16,16,16
170 VDU23,244,0,56,56,16,16,56,54,80
180 VDU23,245,80,80,24,20,36,66,65,65
190 VDU23,246,0,56,56,16,16,56,56,56
200 VDU23,247,56,16,24,24,24,40,40,40
210 FOR T1=300 TO 25 STEP -25
220 FOR X=2 TO 32 STEP 4
230 PRINT TAB(X,16) CHR$(240);TAB(X,17)
    CHR$(241);
240 PRINT TAB(X-1,16) CHR$(32);TAB(X-1,
    17) CHR$(32);
250 FOR T=1 TO T1:NEXT T
260 PRINT TAB(X+1,16) CHR$(242);TAB(X+1
    ,17) CHR$(243);
270 PRINT TAB(X,16) CHR(32);TAB(X,17)
    CHR$(32);
280 FOR T=1 TO T1:NEXT T
290 PRINT(TAB(X+2,16) CHR$(244);TAB(X+2
    ,17) CHR$(245);
300 PRINT TAB(X+1,16) CHR$(32);TAB(X+1,
    17) CHR$(32);
310 FOR T=1 TO T1:NEXT T
320 PRINT TAB(X+3,16) CHR$(246);TAB(X+3
    ,17) CHR$(247);
330 PRINT TAB(X+2,16) CHR$(32);TAB(X+2,
    17) CHR$(32);
340 FOR T=1 TO T1:NEXT T
350 NEXT X
360 PRINT TAB(X-1,16) CHR$(32);TAB(X-1,
    17) CHR$(32);
370 NEXT T1
380 END
```

Fig. 7.3. Animated man program.

man made up from a pair of special symbols one above the other. In this program four pairs of symbols are used, each providing a different picture of the man as he steps forward. This process of adding steps between the basic step movements gives smooth action. It is a relatively crude simulation of walking. By choosing the proper set of positions for the man, more realistic walking action could be achieved.

Smoother movement using linked cursors

Using TAB and PRINT allows us to move our little symbol around the screen but the action is inevitably rather jerky because of the relatively large steps between character positions on the text screen.

The BBC Micro, however, treats text symbols in the same way as graphics ones by shifting the dot pattern of the symbol into the display memory. A very useful byproduct of this is that we can link the text and graphics cursors together and then the text symbol will be displayed at the point indicated by the graphics cursor. This means that we can move the symbol about on the screen one pixel at a time, giving much smoother animation.

Linking of the cursors is achieved by using the command VDU5. We have already used this technique to allow text to be inserted into a graph or chart at a particular position. The next point to note is that the text symbol is set up with its top left corner at the current graphics cursor position. If we are working about the centre of the symbol this can be arranged by offsetting the graphics cursor from the centre of the desired character position in both X and Y directions. This could be done by simply shifting the origin or by allowing for the offset in any cursor position calculations. Here you must remember that the width of the symbols may be 16, 32 or 64 graphics units, although the vertical height of the symbol is always 32.

With the graphics and text cursors linked the symbol can be moved by one pixel at a time to any point on the screen by using the graphics MOVE command. The TAB function is not really applicable in this mode of operation, so now to keep track of an object's position we would need to store the current graphics X,Y position.

Using colour switching and logic functions

Once the VDU5 command has been used the new displayed text will be written in the current graphics colour and its colour is now selected by using GCOL instead of COLOUR.

A useful advantage of this state of affairs is that we can make use of the GCOL logic functions on our new text symbols. We can now erase the symbol by simply printing it twice in the same position. The first PRINT will cause the symbol to appear in the selected colour and the second PRINT statement will cause the symbol to be erased. We can also move our new symbols in front of or behind other objects by using the logical OR, and AND operations provided by GCOL.

The actual process of printing the symbol on the screen may take a short time and the build up might be noticeable. A solution to this is to use colour switching. Before erasing the symbol from its old position its actual colour could be changed using VDU19. After changing its colour instantaneously to say black we can erase the symbol by using GCOL1,ϕ. This will leave unaffected the colour of any other objects overlapped by the symbol.

Using text scrolling to produce motion

When we use the screen to display text and we have filled the page and write in the bottom line you will notice an interesting action occurs. There are two possible ways of finding a space to put the next line of text on the screen. One is simply to start again at the top of the screen and overwrite the top line, then slowly fill the screen again. The other and much more common technique moves all of the lines up by one position to leave a free line at the bottom of the screen. Of course the original top line is now lost. This technique works very much as if we had a scroll of paper and were rolling it past the window formed by the screen, and it is called *scrolling*.

Now when we scrolled the text the whole block of it moved up the screen just like a sort of moving map. Suppose we had two lines of symbols running down the screen to represent the sides of a road and at the top of the screen roughly in the centre of the top row we placed a symbol representing a car. If we scroll the display and write the car symbol back into the new top display row and then draw in a new segment of road at the bottom of the screen we have an animated game of a car running down a road. By adding control of the car

position using the horizontal arrow keys and arranging that the position and width of the road varies in some random fashion as we write in each new segment, we have the basis for a simple game.

Chapter Eight
The Third Dimension

So far all of our graphs and charts have had just two variables X and Y which we have plotted horizontally and vertically on the screen. In the real world, however, we will often find that three variables are involved. The third variable is usually given the name Z. As an example we might want to plot the height of various points in a small area of land. In thise case our X and Y coordinates would represent say length and width and would locate a particular point on the surface of the land. The third term Z will now be the height of the land surface at that point. Here the value of Z will depend upon both X and Y, since a change in either X or Y will take us to another point on the land surface.

Drawing a three-axis graph presents us with some problems. If X and Y are plotted as usual on the screen then the Z coordinates should theoretically be plotted out from the surface of the screen. We could plot Z against X on the screen for different values of Y to give a series of graphs, one for each value of Y. If these are plotted on top of one another the results would be rather confusing. We could use a different colour for each graph, but even this is not very satisfactory.

If we were building a model of the three-axis plot we would probably take each graph of Z against X and stand it up behind the others so that the series of graphs for different Y values are spaced out one behind the other. How can this be done on our display screen?

One solution to displaying such a series of graphs might be to draw them so that the graph for each new value of Y is displaced to the left and up on the screen. This helps to separate the individual graphs for the different values of Y. In effect we are now drawing the Y axis along a sloping line which runs upwards and to the left of the X,Y,Z origin point where X, Y and Z are all equal to 0. This is an improvement on superimposed graphs.

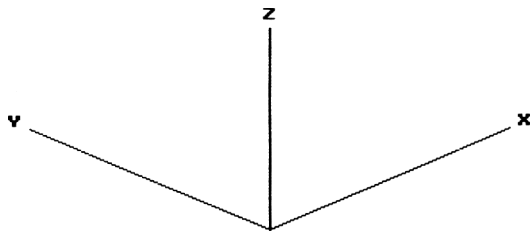


Fig. 8.1. Usual layout for X,Y,Z axes of three-axis graph.

The usual solution to displaying a three-axis plot is to draw the X and Y axes at about 30 degrees to the horizontal axis of the screen, with the Z axis vertical as shown in Fig. 8.1. Now as X increases the plotted point moves upwards and to the right whilst as Y increases the plotted point moves upwards and to the left. Finally, Z just displaces the point vertically on the screen.

Three-axis bar charts

Let us start with a three-axis version of a bar chart. The first step is to choose an origin point where the values of X, Y and Z are all at zero. We will call this point CX,CY on the screen. The next step might be to draw a grid showing the X and Y coordinates in the plane where Z = 0.

To draw the X axis at 45 degrees we need a Y movement which is half the X movement so our screen coordinates for points along the X axis (Y and Z both = 0) will be

```
XS = CX + X
YS = CY + X/2
```

Now consider the Y axis when X and Z are both at 0. Here the line must go up and to the left. The movement being to the left of the origin point CX,CY means that the screen X coordinate must be less than CX. To get the 30 degree angle to the left the X movement is made the same as the Y movement but negative and the Y movement is halved. The screen coordinates here become

```
XS = CX - Y
YS = CY + Y/2
```

If we consider any other point on the Z = 0 plane then the position of

XS,YS will be produced by combining the two results we obtained above to give

```
XS = CX + X - Y
YS = CY + X/2 + Y/2
```

Let us now turn this into a piece of program as shown in Fig. 8.2. This gives us a picture of the X,Y plane of the graph when Z=0 as shown in Fig. 8.3.

```
100 REM X,Y axes for a 3 axis graph.
110 MODE 1
120 GCOL0,1
130 CX = 600:CY=200
140 FOR Y=0 TO 400 STEP 50
150 MOVE CX-Y,CY+Y/2
160 DRAW CX+400-Y,CY+200+Y/2
170 NEXT Y
180 FOR X = 0 TO 400 STEP 50
190 MOVE CX+X,CY+X/2
200 DRAW CX+X-400,CY+X/2+200
210 NEXT X
220 MOVE CX,CY
230 DRAW CX+500,CY+250
240 PLOT0,16,16
250 VDU5
260 PRINT"X";
270 MOVE CX,CY
280 DRAW CX-500,CY+250
290 PLOT0,-48,16
300 PRINT"Y";
310 VDU4
320 END
```

Fig. 8.2. Program to draw X,Y axis for Z = 0.

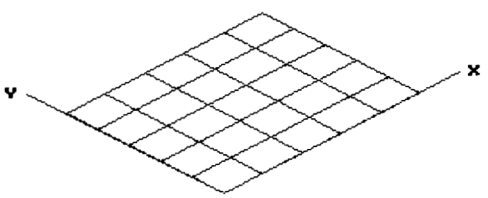


Fig. 8.3. Display of the X,Y axes of the three-axis plot when Z = 0.

The Z term is plotted vertically so it will only affect the YS value and is simply added to YS, so that our final equations become

```
XS = CX + X - Y
YS = CY + X/2 + Y/2 + Z
```

Let us draw a graph in which Z = (X+2*Y)/3. Here the X and Y lane is drawn in one colour and the Z coordinates are drawn in a different colour so that they are easily picked out on the screen. A program to draw such a graph is shown in Fig. 8.4.

```
100 REM Simple 3 axis graph
110 REM for Z=(X+2*Y)/6
120 MODE 1
130 CX=600:CY=100
140 GCOL0,2
150 REM Calculate and plot Z ordinates
160 FOR X = 0 TO 500 STEP 100
170 FOR Y = 0 TO 400 STEP 80
180 Z = (X+2*Y)/6
200 MOVE CX+X-Y,CY+X/2+Y/2
210 PLOT1,0,z
220 NEXT Y
230 NEXT X
240 REM Draw X and Y axes
250 GCOL0,3
260 MOVE CX,CY
270 DRAW CX+550,CY+275
280 PLOT0,32,16
290 VDU5
300 PRINT"X";
310 MOVE CX,CY
320 DRAW CX-450,CY+225
330 PLOT0,-48,16
340 PRINT"Y";
350 MOVE CX-16,CY-16
360 PRINT"0";
370 VDU4
380 END
```

Fig. 8.4. Program to produce a simple three-axis display.

The result is now a pattern of vertical lines looking like a bed of nails where the height of each line indicates the value of Z. To avoid the Z coordinates being merged together the steps on the X and Y axes must be different.

To produce an easier to follow display we could use a different colour for the needles at each value of Y. This will now give us rows

of lines running diagonally across the graph. In a four-colour mode we might use say three colours repeated in sequence, or in mode 2 of course up to seven colours might be used in sequence.

Producing solid bars

Now the three-axis bar chart we have produced has a simple line for each Z coordinate. We can make it look more attractive by making the line into a bar aligned along say the X axis. The changes to the program are quite simple and for our first example the program becomes as shown in Fig. 8.5.

```
100 REM 3 axis bar graph
110 REM for Z=(X+2*Y)/6
120 REM with wide bars
130 MODE 1
140 CX=600:CY=100
150 GCOL0,1
160 FOR X = 500 TO 0 STEP -100
170 FOR Y = 400 TO 0 STEP -80
180 Z = (X+2*Y)/6
200 MOVE CX+X-Y,CY+X/2+Y/2
210 PLOT1,0,z
220 PLOT81,50,25-Z
230 PLOT81,0,Z
240 PLOT3,-50,-25
250 PLOT3,0,-Z
260 PLOT3,50,25
270 PLOT3,0,Z
280 NEXT Y
290 NEXT X
300 GCOL0,3
310 MOVE CX,CY
320 DRAW CX+550,CY+275
330 PLOT0,32,16
340 VDU5
350 PRINT"X";
360 MOVE CX,CY
370 DRAW CX-450,CY+225
380 PLOT0,-48,16
390 PRINT"Y";
400 MOVE CX-16,CY-16
410 PRINT"0";
420 VDU4
430 END
```

Fig. 8.5. Program to draw a three-axis bar chart with wide bars along the X axis.

Lines 21Ø–23Ø draw and fill the bar. The resultant bars are spaced equally apart with the space between them equal to the bar width.

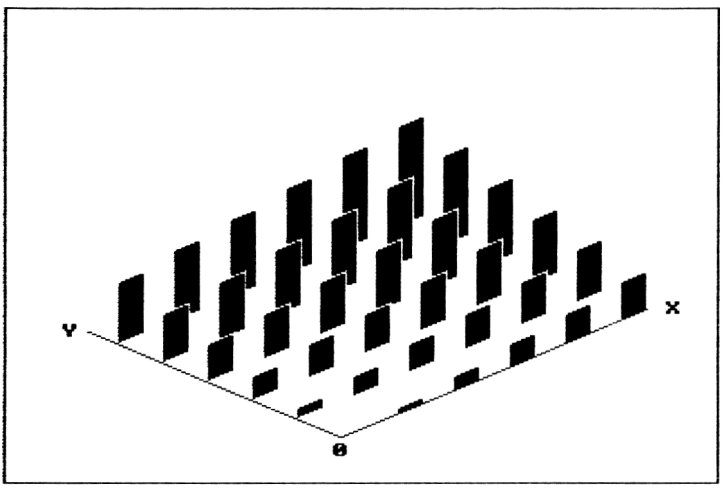


Fig. 8.6. Display of a three-axis chart with wide bars.

The top and bottom of the bars are drawn parallel to the X axis. Note that both X and Y are decremented from maximum to zero so that bars at the rear of the graph are drawn first. Lines 24Ø–27Ø outline the bar in the background colour and this makes the bars in front stand out where they overlap another bar as shown in Fig. 8.6. The width of the bars is easily altered by changing the step size in PLOT statements used to fill and outline the bar.

A further development is to combine these two moves and make an apparently solid bar. The program now becomes as shown in Fig. 8.7 and the result on the screen is shown in Fig. 8.8.

The result is vertical bars with different coloured sides and also a top in a third colour. Mode 1 has been used with the tops of the bars drawn in white. By changing the palette the colours could be set to any desired combination to produce a suitable artistic effect. Mode 2 could also be used to permit the use of more colours.

X,Y bar chart with solid bars

A simple X,Y two-dimensional chart could also be drawn using this type of technique to produce pseudo solid bars. In this case the X axis is tilted to 30 degrees and the background to the bars is drawn in

```
100 REM 3 axis bar graph
110 REM for Z=(X+2*Y)/6
120 REM with solid bars
130 MODE 1
140 CX=600:CY=100
150 REM Plot chart
160 FOR X = 500 TO 0 STEP -100
170 FOR Y = 400 TO 0 STEP -80
180 Z = (X+2*Y)/6
190 REM Draw face of bar
200 GCOL0,1
210 MOVE CX+X-Y,CY+X/2+Y/2
220 PLOT1,0,Z
230 PLOT81,50,25-Z
240 PLOT81,0,Z
250 REM Draw top of bar
260 GCOL0,3
270 PLOT1,-40,20
280 PLOT81,-10,-40
290 PLOT81,-40,20
300 REM Draw side of bar
310 GCOL0,2
320 PLOT81,40,-20-Z
330 PLOT81,-40,20
340 NEXT Y
350 NEXT X
360 REM Draw X and Y axes
370 GCOL0,3
380 MOVE CX,CY
390 DRAW CX+550,CY+275
400 PLOT0,32,16
410 VDU5
420 PRINT"X";
430 MOVE CX,CY
440 DRAW CX-450,CY+225
450 PLOT0,-48,16
460 PRINT"Y";
470 MOVE CX-16,CY-16
480 PRINT"0";
490 VDU4
500 END
```

Fig. 8.7. Program to draw three-axis bar chart with solid bars.

first and filled with colour. The bars are then drawn on top using say half the X step for bar thickness and width. The program for this is shown in Fig. 8.9. Here the bars will appear to be standing out from a blue wall with a red front face and yellow side face on each bar. The bar tops will be white.

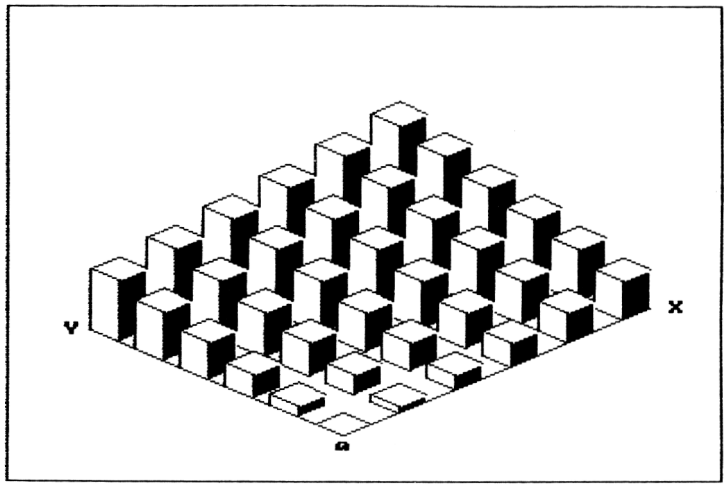


Fig. 8.8. Three-axis bar chart with solid bars.

```
100 REM Bar chart with solid bars
110 MODE2
120 GCOL0,4
130 CX=400:CY=100
140 REM Draw background
150 MOVE CX,CY
160 PLOT1,680,320
170 PLOT81,-680,80
180 PLOT81,680,320
190 REM Draw graph
200 FOR X = 640 TO 0 STEP -80
210 Y=0.6*X
220 MOVE CX+X,CY+X/2
230 REM Draw side of bar
240 GCOL0,3
250 PLOT0,40,-20
260 PLOT81,-40,Y+20
270 PLOT81,40,-20
280 REM Draw front of bar
290 GCOL0,1
300 PLOT1,0,-Y
310 PLOT1,40,20
320 PLOT81,-40,Y-20
330 PLOT81,40,20
340 REM Draw top of bar
350 GCOL0,7
360 PLOT81,-80,0
370 PLOT81,40,20
380 NEXT X
390 END
```

Fig. 8.9. Program for a two-axis bar chart with solid bars.

Surface maps

Instead of drawing vertical lines for the Z coordinates we could draw a picture showing the contours of the surface made up by the tips of the Z coordinates. Here we would take all Z coordinates with Y=0 and draw lines linking them together. Then we repeat the process for each value of Y to give a series of lines running diagonally up to the left. Next we take all Z coordinates for X=0 and again link them with lines. This is repeated for all X coordinates and finally we have a patchwork pattern which will give an impression of the way that Z varies over the X,Y plane. Again we might use colour to pick out the different strips across the plane.

The results obtained using either the vertical Z coordinates or the linked points on the surface will be best if there are a large number of points since this gives a smoother contour to the figure. However, if too many points are used the solid image effect will tend to be lost as some points will overwrite others and produce some confusion. Best results will generally be obtained by arranging the drawing sequence so that it starts at the points toward the back of the X,Y plane, that is with both X and Y at their highest values. Any lines that do overlap will now be overwritten by the line that should appear in front. This is particularly important if different colours are used at the successive X or Y coordinates.

Circular three-axis plots

A rather interesting variation of three-axis plotting is the circular three-axis plot. This produces some rather attractive patterns.

In this version of the three-axis graph the X,Y plane of the chart is made circular and is displayed on the screen as a sort of disc viewed from an angle with circular ridges produced by the Z coordinates.

The technique of plotting uses the quadratic method for drawing a circle but with the X scale set at perhaps two or three times the Y scale. This produces an elliptical figure on the screen. Z values are simply added to the calculated Y coordinates and points are plotted at the tops of the Z coordinates.

The program of Fig. 8.10 gives an example of this type of plot. Function FNZ determines the contour shape of the chart and you could try a variety of functions here. A typical display appears like Fig. 8.11.

If you have a Model A machine use mode 4 for this plot. You

```
100 REM Circular 3D plot program
110 MODE1
120 GCOL0,2
130 VDU19,0,4,0,0,0
140 GCOL0,128:CLG
150 K=5*PI/10000
160 DEF FNA(Z)=10*SIN(K*(X*X+Y*Y))
170 FOR X=-100 TO 100 STEP 2
180 Z1=0
190 Y1=5*INT(SQR(10000-X*X)/5)
200 FOR Y=Y1 TO -Y1 STEP -5
210 Z=INT(80+FNA(SQR(X*X+Y*Y))-.707*Y)
220 IF Z<Z1 THEN 270
230 Z1=Z
240 SX=640+5*X
250 SZ=300+3*Z
260 PLOT69,SX,SZ
270 NEXT Y
280 NEXT X
290 END
```

Fig. 8.10. Program for a circular form of 3D graph.

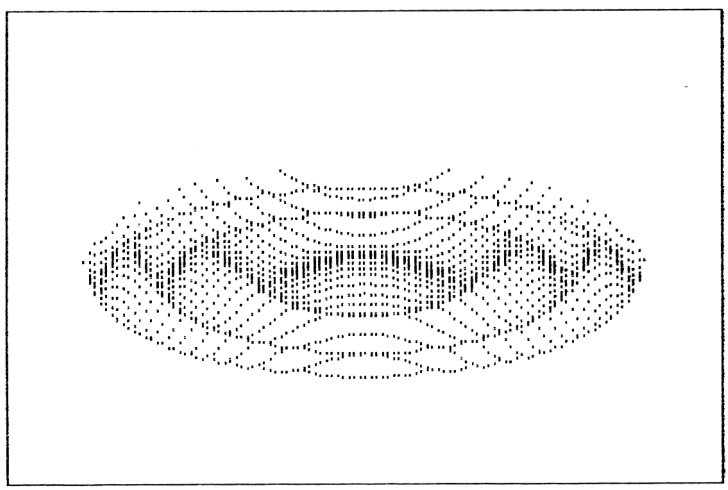


Fig. 8.11. Circular 3D plot produced by the program listed in Fig. 8.10.

could use mode 0 on a Model B and will get very good results from a monitor, but the average televsion receiver tends to give a relatively poor display in this mode.

Perspective drawings

So far we have produced three-axis charts or graphs which give

some impression of depth, but they are in fact what are known as *isometric* drawings. This basically means that a vertical line always has the same scale factor wherever it is on the X–Y plane. This type of drawing is often produced by draughtsmen because it allows correct measurements to be made of the object along all three axes.

To create the illusion of depth an artist uses what is known as *perspective* in his drawings. Artists had discovered many centuries ago the effect that as an object is moved further away from the viewer it appears to get smaller and *vice versa*. They also found that by applying this idea to drawings and paintings they could produce a much more realistic picture. This technique is known as *perspective drawing* and over the years mathematicians have evolved formulae to allow the shape and size of objects to be calculated to give a perspective view. We can also apply this technique in computer graphics.

To see how perspective works, imagine that you are standing on a flat plain and that in front of you is a road that stretches away to the horizon. Although the sides of the road are actually parallel it will appear that the road gets narrower as it approaches the horizon. The cars and trucks travelling along the road also appear to get smaller as they move away from your position toward the horizon. In fact the optical image they produce does get smaller as they move away. If we apply this basic rule to our pictures on the screen we can also produce an illusion of depth despite the fact that our display is really a flat screen.

First, we need to decide on some system of coordinates by which we can measure the positions of points on the objects being viewed and the corresponding points needed to produce the screen image. We shall assume that the X axis runs across from left to right as usual. The Z axis is normally the vertical direction as we had it in our three-axis graphs. This leaves the Y axis and the best arrangement is to have the Y axis along the direction of view.

If you viewed the road across the desert from actual road level (that is with your eye actually at the road surface) the view would be rather uninspiring because every point on the road and the desert would lie along a single line through the X axis. In order to see the road properly we need to be located above it.

Fig. 8.12 shows a side view of the situation where we are viewing the road from an altitude Z. In order to project the image on our flat screen we shall assume that we are looking through a window at distance D.

Suppose we take a point on the road at distance Y1. This will

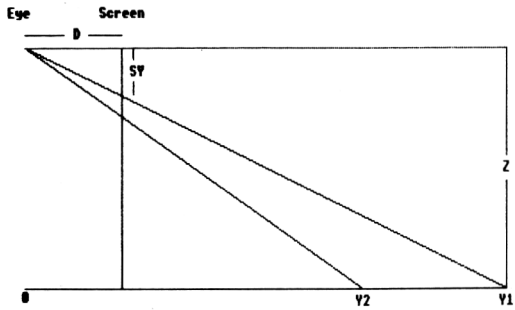


Fig. 8.12. Diagram showing relationship between Y and Z, and SY and D in perspective viewing.

appear to be below the horizon line on the screen by an amount SY. We have assumed here that the horizon is effectively at eye level which it will be if we are looking along a line parallel to the Y axis. Now the small triangle between the screen and eye is of the same shape as the large triangle passing through point Y1. This means that all of the sides of the two triangles have the same proportions, so we can say that

```
SY/D    =    Z/Y1
```

and rearranging this we get

```
SY = Z*D/Y1
```

Now you will note that the size of the image on the screen is inversely proportional to the distance Y1. If we took another point on the road at a different distance Y2 then it would produce a different line length on the screen, giving a new value for SY of

```
SY = Z*D/Y2
```

Now suppose there were a series of equal length lines drawn along the centre line of the road. Each line will produce a short vertical line on the screen and as the point on the road gets farther away the image produced on the screen by each line gets shorter.

If we drew a similar view of the road looking down on it we would find that the same basic formula applies for the SX image size on the screen which represents the road width. As Y1 becomes greater the width of the image on the screen decreases according to the formula

```
SX = W*D/Y1
```

where W is the width of the road.

Vertical objects alongside the road will also produce images that follow this general rule of being inversely proportional to distance Y.

Let us start with the road. First, we need to draw a horizon line across the screen, since this acts as a visual reference. Now if we draw lines from the bottom two corners of the screen to a point halfway across the horizon line we have a road. Suppose the road is 25 feet wide and we are viewing a 14 inch TV screen from, say, a distance of 4 feet. The TV screen will be 1 foot wide and this is equal to SX. D will be 4 and X will be 25. Now

```
SX = D*X/Y = 4*25/Y =1
```

therefore

```
Y = 4*25 or 100 feet
```

The scaling factor for our screen drawing SCX can now be worked out since when Y=100 and X=25 and D=4 then the screen value for X must be 1280. If we rewrite our equation with a multiplier SCX we get

```
SX = SCX*X/Y
```

and inserting our calculated values we shall get SCX = 5120, so to plot any point we calculate its X value from

```
SX = 5120*X/Y
```

We can apply a similar process to calculate the Y scale factor, assuming that our viewpoint is at a height of 10 feet and that for a distance of 100 feet the point plotted is at the bottom of the screen. We shall assumed that the horizon line is at Y = 500 on the screen. From this

```
SY = SCY*10/100 =500
```

and

```
SCY = 5000
```

so to calculate the Y points on the screen we would use

```
SY = 5000*Z/Y
```

Note that here we have incorporated the distance D into the scale factor, since D is a constant at 4 feet.

To plot a point on the road itself Z=10 and the actual Y value for

use in the PLOT instruction will be 500 −SY, since as we get closer the point moves down the screen. If there is a vertical pole then to plot the top of the pole we subtract the height of the pole from 10 to obtain the value for Z. Thus a 10 foot high pole will always produce a Y value of 500 and would line up with the horizon, since we are actually looking along a line 10 feet above the road. If the pole is higher than 10 feet the value of SY is negative and the top of the pole is above the centre line of the screen.

To draw a perspective view of a road with, say, trees alongside, you will need to set up a table of values of height and distance for the trees and then their base and top points can be calculated using

$$SX = 64\emptyset + 512\emptyset * X/Y$$
$$SY = 5\emptyset\emptyset - 5\emptyset\emptyset\emptyset * (1\emptyset - Z)/Y$$

where SX and SY are the screen plotting coordinates, X is the distance measured from the centre of the road with positive values to the right, and Z is the height of the object. For road markings Z will be 0.

Fig. 8.13 shows a program listing to draw a perspective view of a road, and Fig. 8.14 shows the result on the screen.

```
100 REM Program to draw picture of road
110 MODE 2
120 REM Draw sky
130 GCOL 0,6
140 PLOT0,0,512
150 PLOT1,1279,0
160 PLOT81,0,511
170 PLOT1,-1279,0
180 PLOT81,0,-511
190 REM Draw desert
200 GCOL0,3
210 PLOT1,1279,0
220 PLOT81,0,-512
230 PLOT1,-1279,0
240 PLOT81,0,512
250 MOVE 640,512
260 REM Draw road
270 GCOL0,0
280 PLOT1,639,-512
290 PLOT81,-1279,0
300 SCX=5120:SCY=5000
310 Z=10:W=1:WR=25/2
320 REM Draw road markings
```

```
330 GCOL0,7
340 FOR Y1=100 TO 2000 STEP 100
350 SY=512-SCY*Z/Y1
360 X=SCX*W/Y1
370 SX=640-X
380 PLOT4,SX,SY
390 SX=640+X
400 PLOT5,SX,SY
410 Y2=Y1+50
420 SY=512-SCY*Z/Y2
430 X=SCX*W/Y2
440 SX=640-X
450 PLOT85,SX,SY
460 SX=640+X
465 PLOT85,SX,SY
470 NEXT Y1
480 REM Draw trees
490 FOR Y1=120 TO 1000 STEP 100
500 H=15:HT=5
510 SY=512-SCY*Z/Y1
520 X=SCX*WR/Y1
530 SX=640-X
540 MOVE SX,SY
550 PROCTREE
560 SX=640+X
570 MOVE SX,SY
580 PROCTREE
590 NEXT Y1
600 END

1000 DEFPROCTREE
1010 GCOL0,1
1020 SH=SCY*H/Y1
1030 PLOT1,0,SH
1040 SH=(H-HT)*SCY/Y1
1050 PLOT0,0,0
1060 SW=2*SCX/Y1
1070 GCOL0,2
1080 PLOT81,-SW,-SH
1090 PLOT81,2*SW,0
1100 ENDPROC
```

Fig. 8.13. Program to draw a perspective view of a road.

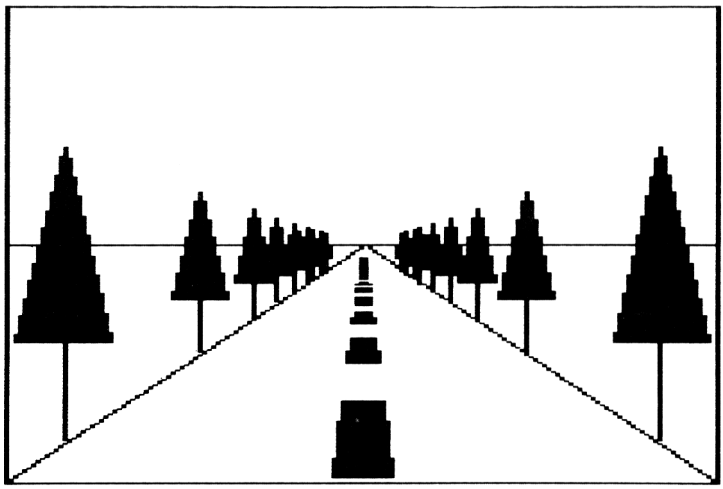


Fig. 8.14. Perspective view of a road produced using the program in Fig. 8.13.

Wire frame models of solid objects

When we come to drawing perspective images of three-dimensional objects we use an array of X, Y, Z coordinates to define points on the object. When the image is drawn these points are linked together by lines to produce an outline of the edges of the object. If we were drawing a rectangular block then the reference points would be the eight corners of the block and the linking lines represent the edges of the block. This form of drawing is called a *wire frame image* because it is as if we made up the object by building a wire frame marking out its edges.

For viewing the object along its Y axis we can use the same basic proportional technique that was used for the perspective view of a road. If we want to be able to move all around our object so that we can view it from all directions then the equations become more complex. We shall now go on to look at this situation.

All round viewing of objects

To produce a perspective view of a wire frame object from any point around the object becomes fairly complex. First, we need to define the object as a set of X,Y,Z coordinates relative to its own centre. We shall also need a set of information which shows where the edges are

relative to the X, Y, Z points. In addition we will need some information to tell the computer whether it must use MOVE or DRAW when tracing out the image from point to point on the screen. This can be arranged by creating three arrays of data describing the object. The X, Y and Z coordinates of the object in its descriptive array are all measured relative to a point at the centre of the object itself. Next we have a viewer at some point XV, YV, ZV relative to the object.

The first stage in the process is to translate the X,Y,Z coordinates of the object so that the viewer is placed at the origin of the X,Y,Z axes. This is the point where X, Y and Z are all 0. This translation process is quite straightforward since it simply involves subtracting XV,YV,ZX from X,Y and Z respectively to give a new set of values for the X,Y and Z coordinates of the points on the object.

At this point we have assumed that the viewer is absolutely level. For all round viewing we can assume that the line of sight of the viewer has three angular components, which we shall call *heading, pitch* and *roll.* To understand these it is useful to imagine that you are flying in an aeroplane. The Y axis of the aeroplane is assumed to be along the fuselage and the X axis along the wings.

Heading is the direction of view of the viewer measured relative to the Y axis. A change in heading is equivalent to turning to the left or right. This is effectively a rotation of the aircraft around its Z or vertical axis. Next there is pitch, which shows wnether you are looking above or below the horizontal. This shows whether the aircraft has a nose down or nose up attitude and is equivalent to rotating the plane around the line along its wings. This is effectively a rotation about the X axis of the aircraft. Finally, there is roll which indicates if your view is inclined to the right or left. This is like rotating the plane around a line along its fuselage, which is what happens when an actual aircraft rolls. Here the rotation is around the Y axis of the aircraft.

After shifting the origin of the X, Y, Z map we will have the position where the viewer and the object being viewed are on the Y axis but the viewer is in fact looking at a point away from the Y axis by his heading angle. This could place the object off the screen. In order to place the object in the centre of the field of view and at its correct orientation we will need to rotate its points about the three axes to correct for the heading, pitch and roll of the viewer. This is done in three stages.

First, the points are rotated around the Z axis until the viewer is looking directly through the centre of the object. This is done by

rotating all of the points on the object through the heading angle of the viewer. The rotation equations are the same as those used in Chapter 2, but now they are applied to X,Y and Z. In this first roation Z1 will be equal to the original Z value since we are rotating around the Z axis itself. The new X and Y values become

```
X1=X*COS(H)-Y*SIN(H)
Y1=X*SIN(H)+Y*COS(H)
```

In the second stage of calculations these X1,Y1,Z1 values are rotated by the pitch angle to produce a new set of values X2,Y2,Z2. In this rotation the X values are unchanged since the points are being rotated around the X axis. The second set of values X2, Y2, Z2 are finally given a further rotation by the roll angle to produce the coordinates X3,Y3,Z3. In this last rotation, which is around the Y axis, the Y2 values will be unchanged.

Having produced these rotated coordinates we have reached roughly the position we were in with the road view. It remains to project the points on to the screen and this basically involves dividing X3 and Z3 respectively by Y3 to obtain the screen coordinates XS and YS.

The whole process of rotation and projection is carried out for each coordinate point on the object and then the appropriate lines are plotted between the points to form the picture on the screen.

```
100 REM All round viewing of a block
110 MODE 1
120 SCX=50:SCY=50
130 DIM V(50,3),E(100),C%(100)
140 REM Read object data
150 READ NV
160 FOR P=1 TO NV
170 READ V(P,1),V(P,2),V(P,3)
180 NEXT P
190 READ NE
200 FOR J= 1 TO NE
210 READ E(J),C%(J)
220 NEXT J
300 REM Set viewer position
310 D=80:P=PI/16:R=0:H=PI/4
320 PROCMF
330 XV=-D*CP*SH
340 YV=-D*CP*CH
350 ZV=-D*SP
400 REM Project object on screen
410 FOR J=1 TO NE
420 N=E(J)
```

```
430 X=V(N,1):Y=V(N,2):Z=V(N,3)
440 PROCTRANS
450 GCOL0,C%(J)
460 DRAW XS,YS
470 NEXT J
500 INPUT TAB(0,1)"Another view? Y/N "A$
510 IF A$="Y" OR A$="YES" THEN 540
520 IF A$="N" OR A$="NO" THEN 600
530 PRINT TAB(0,1):PRINT:GOTO 500
540 INPUT TAB(0,1)"Heading (deg)= "H
550 INPUT "Pitch (deg)= "P
560 INPUT "Roll (deg)= "R
570 H=H*PI/180:P=P*PI/180:R=R*PI/180
580 CLG
590 GOTO 410
600 END
1000 REM Multiplier factors
1010 DEFPROCMF
1020 CH=COS(H):SH=SIN(H)
1030 CP=COS(P):SP=SIN(P)
1040 CR=COS(R):SR=SIN(R)
1050 MF1=CH*CR-SH*SP*SR
1060 MF2=-SH*CR-CH*SP*SR
1070 MF3=CP*SR
1080 MF4=SH*CP
1090 MF5=CH*CP
1100 MF6=SP
1110 MF7=-CH*SR-SH*SP*CR
1120 MF8=SH*SR-CH*SP*CR
1130 MF9=CP*CR
1140 ENDPROC
1200 DEFPROCTRANS
1210 REM Translate viewer position
1220 X=X-XV:Y=Y-YV:Z=Z-ZV
1230 REM Rotate view of object
1240 X3=MF1*X+MF2*Y+MF3*Z
1250 Y3=MF4*X+MF5*Y+MF6*Z
1260 Z3=MF7*X+MF8*Y+MF9*Z
1270 REM Project points to screen
1280 XS=INT(640+SCX*D*X3/Y3)
1290 YS=INT(512+SCY*D*Z3/Y3)
1300 ENDPROC
2000 REM Number of points
2010 DATA 8
2020 REM X,Y,Z coordinates
2030 DATA 5,-3,4
2040 DATA -5,-3,4
2050 DATA -5,-3,-4
2060 DATA 5,-3,-4
2070 DATA 5,3,4
```

```
2080 DATA -5,3,4
2090 DATA -5,3,-4
2100 DATA 5,3,-4
2110 REM Number of edges
2120 DATA 16
2130 REM Edge and colour data
2140 DATA 1,0,2,1,3,1,4,1
2150 DATA 1,1,5,3,6,2,7,2
2160 DATA 8,2,5,2,2,0,6,3
2170 DATA 3,0,7,3,4,0,8,3
```

Fig. 8.15. Perspective view of a block.

The program listing given in Fig. 8.15 carries out this process on an object which is a wire frame model of a simple block. The front and rear faces of the block are plotted in different colours on the display to make it a little easier to interpret the picture. By inputting the direction of view in terms of heading, pitch and roll as seen from the viewer's position, the corresponding view of the object is displayed. In effect we are rotating the object itself to present the correct view. Fig. 8.16 shows a typical view on the screen.

Aircraft landing sequence

Now let us look at the problems of animation in a perspective view. As you will have seen, the calculations needed to produce a perspective view are very slow when BASIC is used. One solution is

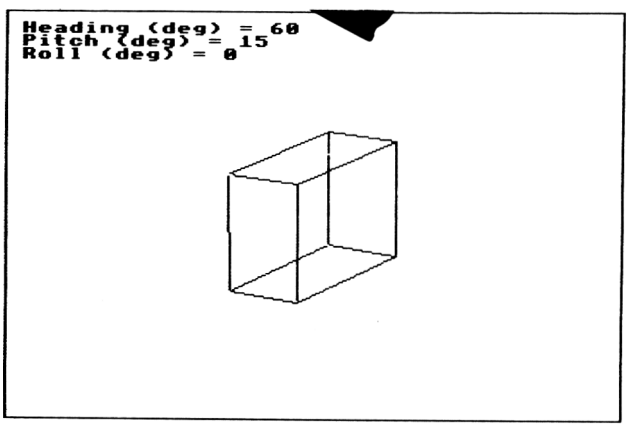


Fig. 8.16. Display of a wire frame object in perspective.

to write the program in machine code, and this has been done in 3D simulation games. An alternative is to simplify the problem to make the calculations faster.

Let us assume that we are in an aircraft coming in to land on a runway and we are going to display the view from the aircraft. The aircraft is assumed to be flying with a level attitude above the ground and the height varies as it approaches the runway. To simplify the calculations we shall assume that the plane does not deviate from a line along the centre of the runway.

If we consider two points, one at each end of the runway, then initially when the distance is large the two points will be close together near the horizon. As we move closer the points will separate and move down the screen with the nearer point being lower. As in the case of the road the Y position of each point on the screen is proportional to the ratio A/D for that point. Now we would run into problems if D were zero, since this produces a 'divide by zero' error. To avoid this it has been assumed that the angle of view from the cockpit is limited so that the nearest point that is displayed is at a distance D equal to altitude A. By carrying out a test for this before calculating the Y term we can avoid problems. When we consider points off the centre of the runway we can apply the same ratio technique.

First we must set up a table of points representing the runway, all in terms of real distances relative to the centre line at the far end of the runway. To draw the picture each point X(N),Y(N) in the real world is processed to produce a new point SX(N),SY(N) for the screen display.

In the display procedure a second set of coordinates LX(N),LY(N) is used to represent the previous position of the aircraft. First the image is erased by drawing the LX,LY picture in the background colour. The new picture is then drawn using the points SX,SY, and finally the LX,LY points are made equal to the SX,SY points ready for the next picture to be drawn.

The calculations of aircraft position are governed by a simple aerodynamic calculation which takes into account the pitch angle P and the engine power E. These variables are controlled by the arrow keys on the keyboard. Here the vertical arrows control the plane's pitch with the up arrow being equivalent to pushing the joystick forward, which increases both speed and rate of descent. The up-down arrows move the pitch in steps of 5 degrees up to a maximum of 30 degrees. Engine power is controlled by the left/right arrows, with the right arrow giving increased power. Power is displayed as a

percent value on the screen and changes in power level will affect both rate of climb and speed.

```
100 REM Aircraft landing program
110 MODE1
120 *FX4,1
130 VDU28,0,4,39,0
140 GCOL0,3
150 DIM X(12),Y(12)
160 DIM SX(12),SY(12),LX(12),LY(12)
170 SCX=1000:SCY=1000
180 CX=640:CY=500
190 FOR N=1 TO 12
200 READ X(N),Y(N)
210 LX(N)=0:LY(N)=0
220 NEXT N
230 DATA 25,4000,50,3800,100,3600
240 DATA 50,3200,50,2800,50,2400
250 DATA 50,2000,50,1600,50,1200
260 DATA 50,800,50,400,100,0
270 K1=10:K2=5:K3=.95:K4=.05
280 K5=-0.5:K6=.1:K7=100:K8=50
290 K9=.9:K10=.1:K11=.2:K12=.05
300 K13=.4:K14=-20:VS=75
310 D=19000+RND(2500):E=50:P=0
320 V=125:A=(D-3000)*.06
330 CL=-400:DA=-8
340 PROCDRAW
350 PROCAERO
360 IF A<10 THEN 390
370 PROCINSTR
380 GOTO340
390 CLS
400 PRINT"You have ";
410 IF V>150 OR CL<-200 THEN PRINT
    "CRASHED";
420 IF V<151 AND CL>-199 THEN PRINT
    "LANDED";
430 IF D>0 AND D<3800 THEN PRINT" on
    the runway"
440 IF D>3800 THEN PRINT INT(D-3800);
    "  feet short of the runway"
450 IF D<0 THEN PRINT INT(-D);" feet
    beyond the runway"
460 PRINT "Do you want to try again -
    Y/N ? ";
470 B$=INKEY(100)
480 IF B$="Y" THEN 310
490 IF B$="N" THEN 510
500 GOTO 470
```

```
1000 DEFPROCDRAW
1010 FOR N=1TO12:Y1=D-Y(N):IF Y1<A Y1=A
1020 SY(N)=SCY*A/Y1:SX(N)=SCX*X(N)/Y1:
     NEXT N
1030 FOR N=1TO12:PLOT4,CX-LX(N),CY-LY(N
     ):PLOT3,2*LX(N),0
1040 MOVE CX-SX(N),CY-SY(N):PLOT1,2*SX(
     N),0:NEXT N
1050 MOVE CX-LX(12),CY-LY(12):PLOT7,CX-
     LX(3),CY-LY(3)
1060 PLOT0,2*LX(3),0:PLOT7,CX+LX(12),C-
     LY(12)
1070 MOVE CX-SX(12),CY-SY(12):DRAW CX-S
     X(3),CY-SY(3)
1080 PLOT0,2*SX(3),0:DRAW CX+SX(12),CY-
     SY(12)
1090 FOR N=1TO12:LX(N)=SX(N):LY(N)=SY(N
     ):NEXT N
1100 ENDPROC

2000 DEFPROCAERO
2010 K=INKEY(20)
2020 IF K=136 THEN E=E-K1
2030 IF K=137 THEN E=E+K1
2040 IF K=138 THEN P=P+K2
2050 IF K=139 THEN P=P-K2
2060 IF E>100 THEN E=100
2070 IF E<0 THEN E=0
2080 IF P>30 THEN P=30
2090 IF P<-30 THEN P=-30
2100 V=K3*V+K4*(K5*P+K6*E+K7)
2110 CL=K8*DA
2120 A=A+DA
2130 DA=K9*DA+K10*(K11*E+K12*V+K13*P+K
     14)
2140 D=D-V
2150 IF V<VS THEN DA=-40
2160 ENDPROC

3000 DEFPROCINSTR
3010 CLS
3020 PRINT TAB(0,1)"ALT = ";INT(A);TAB(
     6);"DIST. FT. ";INT(D-3600)
3030 PRINT"POWER%= ";INT(E);"   SPEED K
     TS.= ";INT(V)
3040 PRINT"CLIMB RATE FT/MIN= ";INT(CL);
3050 COLOUR 2
3060 PRINT TAB(30,3)"          "
3070 IF D<4000 THEN 3100
3080 IF A>.065*(D-3000) THEN PRINT TAB(
     30,3);"TOO HIGH"
3090 IF A<.055*(D-3000) THEN PRINT TAB(
     30,3);"TOO LOW"
3100 COLOUR3:ENDPROC
```

```
3000 DEFPROCINSTR
3010 CLS
3020 PRINT TAB(0,1)"ALT = ";INT(A);TAB(
     6);"DIST. FT. ";INT(D-3600)
3030 PRINT"POWER%= ";INT(E);"   SPEED K
     TS.= ";INT(V)
3040 PRINT"CLIMB RATE FT/MIN= ";INT(CL);
3050 COLOUR 2
3060 PRINT TAB(30,3)"          "
3070 IF D<4000 THEN 3100
3080 IF A>.065*(D-3000) THEN PRINT TAB(
     30,3);"TOO HIGH"
3090 IF A<.055*(D-3000) THEN PRINT TAB(
     30,3);"TOO LOW"
3100 COLOUR3:ENDPROC
```

Fig. 8.17. Aircraft landing program.

Pitch, engine power, height, distance to the end of the runway and rate of climb are displayed at the top of the screen. Note that the plane does not respond immediately to a new power or pitch setting but gradually changes its attitude because there is an integration effect built into the equations to give a better simulation of a real plane.

The lower the plane is as it approaches the runway the more effective the perspective effect will be. In this simple demonstration the plane starts at varying height and distance for each run.

The plane will stall at speeds less than 75 knots and the rate of descent will then increase very rapidly. A descent rate of about 500 feet per minute is required on the approach and a warning message will tell you if you are too high or too low relative to a 3.5 degree glide path. At about 5000 to 6000 feet from the end of the runway you should level out by increasing the pitch angle to give a descent rate of about 100 to 150 feet/min for landing. Power setting of about 20% is quite good for landing where the speed should be less than 150 knots and the rate of descent less than 200 feet/min.

The program shown in Fig. 8.17 is a relatively basic simulation program intended to show the possibilities of animation with perspective. You could alter the characteristics of the runway by changing the data in lines 23Ø to 26Ø. The characteristics of the plane can be altered by changing the constants K1 to K14. It would also be possible to rearrange the program to allow a take-off from the runway and then after a short flight a landing on the same runway. The main problem, however, is that the more complex the program becomes the slower will be the steps between displayed pictures and eventually the real time operation will be lost.

Chapter Nine
Making Sounds

So far we have been concerned with producing pictures on the screen and we have in effect the equivalent of a silent film in the cinema. The addition of sound effects or music would greatly enhance the overall results, particularly when we are dealing with a game type program. In fact the BBC Micro has a very versatile sound producing capability and we could in fact use it as a form of musical instrument, where the sound performance would be more important than the picture on the screen.

Early personal computers were fitted with very simple sound generators which could produce a 'beep' to attract the user's attention in much the same way as the bell on a typewriter. Most modern types, however, have more sophisticated sound generators which permit a range of different tones to be produced and perhaps the amplitude to be controlled. A few machines, such as the BBC Micro, provide several independent sound generation channels so that two or more notes may be produced simultaneously. These more complex sound generators allow the computer to produce a wide range of sound effects which are useful in games programs and will also permit music to be produced by the computer.

The BBC Micro uses a special sound generating chip called the SN76489, made by Texas Instruments. This chip, or others like it, has been used in many of the newer personal computers. There are in fact four separate sound generators in the SN76489 chip, each of which may be controlled independently of the others by the computer itself. Three of the four sound channels provide a tone output which can be programmed for pitch (frequency), loudness (amplitude) and duration. The fourth channel is rather different, because it produces a noise output which may also be programmed to give various kinds of noise. You might wonder why we should want a noise generator at all, but as we shall see later, noise is often an important component in producing sound effects and this can be

particularly useful in providing the sounds required for games programs.

Although the sound generator chip itself is quite versatile, an important feature of the BBC Micro is the software available to control the actual sounds that are produced. A particularly useful feature is the *envelope control* which allows us to control the loudness and pitch of the sounds, as they are being generated. This allows us to control the way in which the sound builds up or dies away and enables us to imitate the 'voices' of musical instruments and to produce special sound effects without having to use a lot of computer instructions. We shall be looking more closely at this later. It is also possible to synchronise the sound channels so that we can play several notes at once, producing a chorus effect. The possibilities of the sound generation of the BBC Micro seem to be almost limitless and all we can hope to do in this book is show how the various parts work and give some leads so that the reader can experiment for himself and discover what can be achieved.

Surprisingly enough for such a complex sound capability, there are only two BASIC commands which control the sound generation in the BBC Micro. These are SOUND, which governs the basic pitch, loudness and duration of sounds and actually causes them to be produced, and ENVELOPE, which sets up the way in which the loudness and pitch will vary as the sound is being generated. In fact you could have a lot of fun producing sounds without using the ENVELOPE command at all. Although there are only two basic commands, each has a wide range of options and the ENVELOPE command may seem to be particularly complicated when it is first encountered, but as we shall see, it can be broken down into simple terms. Let us now explore these commands and see how they can be applied to the business of producing sounds from the BBC Micro.

The sound command

Let us start by taking a look at the SOUND command which governs the actual production of a sound from the loudspeaker on the BBC Micro. Using this command alone it is possible to produce a wide variety of sound effects. The command itself has four parameters which control the actual sound that is generated. The basic form of the command is

```
SOUND  C, A, P, D
```

where

C is the channel number
A is the sound amplitude
P is the pitch of the note
and D is the duration of the sound.

The four channels are numbered 0, 1, 2 and 3. Channels 1, 2 and 3 are identical but separate channels for generating tones. Channel 0 is rather different in operation and it generates various kinds of 'noise' sounds. We shall look at this noise channel more closely in a moment. For a start let us experiment with using the tone channels.

The channel, or channels, which produce signals to drive the loudspeaker are selected by the first parameter C. If we want to select channel 1 then C is made equal to 1, so that the command becomes

```
100 SOUND 1,A,P,D
```

Similarly to select channel 2 or 3 we would simply put C equal to 2 or 3.

As we shall see later, the first parameter C also performs a number of other functions, but the moment we can look upon it as simply a channel selection code.

Controlling the loudness

An important characteristic of any sound is its loudness or amplitude, and this is governed by the value of A which can range from +15 to −15. Putting A=0 as we might expect produces no sound at all. This may seem a bit pointless but as we shall see later the sound generator will effectively just produce a pause in its output governed by the duration D.

For some strange reason the loudness actually increases as the value of A becomes more negative until at −15 the loudest output is produced. Let us try an experiment with the following small program:

```
100 FOR A = 0 TO -15 STEP -1
110 SOUND 1, A, 101, 20
120 SOUND 1, 0, 101, 1
130 NEXT
140 GOTO 100
```

This should produce a single note with its volume rising in steps

from zero to the maximum value and will repeat indefinitely. Here the second SOUND statement (line 12ø) shows the use of zero amplitude and merely provides a short pause to separate successive notes. Use ESCAPE to stop the program. Note here that the volume remains steady whilst each note is sounded and only changes to a new level when the next SOUND command is executed.

Positive values for A do not directly control the loudness of the sound produced. These values are in fact used in conjunction with the ENVELOPE command which allows the loudness of the sound to be varied as the note is being played.

Making different notes

The second important characteristic of any tone or note is its frequency or pitch. All sounds are produced by a variation in air pressure which for a pure tone follows a sinusoidal waveform. The frequency or pitch is the number of complete cycles of pressure variation that occur in one second. This may be expressed either as cycles per second (c/s) or in hertz (Hz), but both have the same numeric value. The normal range of sounds that we can hear extends from about 30 Hz, which is a low frequency hum, up to about 15,000 Hz (15 kHz) which is a very high pitched whistle. Most of our basic sounds of speech and music, however, tend to fall in the range from about 100 Hz to 4 kHz.

The parameter P controls the pitch of the tone produced by the sound generator. The value of P can extend from 0 to 255 and should be an integer, so the P variable will normally be of the form P%. A value of 0 gives the lowest frequency tone and the pitch increases as the value of P is increased.

A value of 1 for P gives a frequency of about 122 cycles per second, equivalent to roughly the note B an octave below middle C on the musical scale. Middle C on a musical scale has a frequency of 261.6 Hz and needs a P value of 52. At the upper end of the range a value of 255 for P gives a note of about 4200 Hz.

To see the range of tone pitch that can be produced, try running the following short program:

```
100 FOR P = 0 to 255
110 SOUND 1, -15, P, 20
120 SOUND 1,0,P,2
130 NEXT
140 GOTO 100
```

This should produce a steadily rising note and shows the effect of varying the parameter P. The second SOUND statement at line 12Ø merely provides a short silent period between successive notes to prevent them from just running into one another. The pitch produced remains constant throughout the period when the note is produced and only steps up to a new value when the next SOUND command is executed. We shall see later that it is possible to vary the pitch during the sounding of a note by using the ENVELOPE command and this can completely alter the character of the sound being produced.

Controlling the length of sounds

Apart from setting the loudness and pitch we also need to be able to control the length of time for which the sound is produced. This is known as the sound duration and is governed by the value of D, the last parameter in the SOUND command. D is an integer number with a value from 1 to 255 and in fact represents the number of $\frac{1}{20}$ second intervals that the sound will last for. The timing here is accurate and is independent of the computer program.

Once the SOUND command has been executed the computer can go on and do something else, but the sound will still continue to be produced until the duration D has expired. So by using D we can produce notes varying in length from $\frac{1}{20}$ second up to about $12\frac{1}{2}$ seconds.

We can in fact turn on a sound channel for an indefinite time by using a value of −1 for the parameter D. Now even if the program ends the sound would continue. After a SOUND command with D = −1 we must either issue another SOUND command to end the note or press ESCAPE to abort the whole program action.

Let us now see how the basic scheme works. Enter the following small program:

```
10 SOUND 1,-15,100,100
40 FOR T=1TO200:NEXT:GOTO10
```

When run, this will produce a continuous series of 'beeps'. Now add the lines

```
20 SOUND 2,-10,160,75
30 SOUND 3,-6,200,200
```

and rerun the program. You should now have a shorter and lower

pitched beep and a longer higher pitched beep and these two new tones will be quieter than the first.

Because the three sound channels are independent we can in fact sound up to three notes at once to produce richer sounds or to produce musical chords.

The noise channel

Channel 0 is unlike the other three channels because it generates various 'noise' types sounds instead of tones. The SOUND command to control channel 0 is basically the same as for the other channels but there are some differences in the way the pitch parameter is used.

Noise is made up from a mixture of sounds with random pitch and amplitude. When we consider this type of sound the pitch value has little meaning, except in general terms such as high, medium or low pitch. In fact for channel 0 there are only eight possible values for P, which run from P=0 to P=7. Instead of controlling pitch the P term now selects the type of noise produced.

P values 0, 1 and 2 produce periodic noise of high, medium and low pitch respectively. This sounds like a rather raspy sort of note similar to that produced by a buzzer.

P values of 4, 5 and 6 are similar to 0, 1 and 2 except that the noise produced is *white noise*. Whereas the periodic noise has a regular frequency component and produces a rather rough note, white noise contains a wide mixture of random frequencies and amplitudes. The effect of setting P=4 is to produce a high pitched hiss which sounds like escaping steam. The values of 5 and 6 for P produce lower frequency noise similar to that produced by a radio or television receiver when it is not tuned to a station.

As an example of the use of white noise we can imitate the effect of a steam engine with the program listed in Fig. 9.1. By adjusting the

```
100 REM Steam engine effect
110 FOR N=4 TO 160
120 D=12-N/4
130 IF D<2 THEN D=2
140 SOUND0,-15,4,D
150 FOR T=1 TO D*120:NEXT T
160 NEXT N
170 END
```

Fig. 9.1. Program to show effect of using white noise.

```
100 REM Noise linked to Channel 1 pitch
110 CLS
120 PRINT
130 PRINT"Periodic noise (P=3)"
140 FOR P=1 TO 200
150 SOUND1,0,P,5
160 SOUND0,-15,3,5
170 NEXT P
180 PRINT
190 PRINT"White noise (P=7)"
200 FOR P=1 TO 200
210 SOUND1,0,P,5
220 SOUND0,-15,7,5
230 NEXT P
240 END
```

Fig. 9.2. Program showing effect of noise linked to channel 1 pitch.

timing rather more realistic results might be produced but this serves to give a general idea of this type of sound effect.

When we put P=3 we get a rather interesting effect because although we still get noise its pitch is governed by the pitch being used on channel 1. So here we have a variable pitch capability of sorts. In the short program listed in Fig. 9.2 the effect of this is tried by running channel 1 up through its pitch range but using an amplitude of 0 so that only the noise channel is actually heard.

At the lower frequency end the effects are similar to those of a car engine and this could well be used to produce car type sound effects for say a road race type of game. Here the pitch on channel 1 would be related to the engine speed to get the correct effects as the car speeds up or slows down. As the pitch of channel 1 becomes higher we find that we now have notes of much lower frequency than was possible on the other sound channels. This provides a possible means of extending the range of sounds down into the bass region.

Finally we can use P=7 which produces a white noise related to the pitch value used on channel 1.

Shaping up the sound

So far we have, by using the SOUND command, been able to produce a variety of sounds of different pitches and amplitudes. In all of these sounds, however, the pitch and amplitude have remained constant throughout the duration of the sound. In real life the amplitude and

pitch of sounds actually change as the sound is produced and it is these changes which give many sounds their own particular character.

At this point we shall introduce the ENVELOPE command which tends to set the BBC Micro apart from many other personal computers, because it allows us to control the character of the sounds that we can produce with the SOUND command. A point to note here is that the ENVELOPE command by itself does not cause any sound to be produced and its sole function is to modify the sounds actually generated by a SOUND command.

ENVELOPE is a very complex and versatile command and its possibilities seem to be limited only by the imagination and ingenuity of the user. Here we shall explain how it works and try to show some of the things that you can do by playing with the values of the ENVELOPE command parameters. Although at first this command may seem to be very complicated it can be broken down into a set of relatively straightforward functions.

The actual command has a total of 14 different parameters which can be set up, and to avoid confusion we shall use the same abbreviations for these as in the BBC Micro *User's Manual*. The format for ENVELOPE is

```
100 ENVELOPE N,T,PI1,PI2,PI3,PN1,PN2,
    PN3,AA,AD,AS,AR,ALA,ALD
```

Let us look at them in a little more detail.

The first parameter N is quite straightforward because it is the envelope number. We can in fact set up as many as 15 different envelopes and these are identified by this envelope number. When we wish to use a particular envelope to control a sound the amplitude A term of the SOUND command is set to the envelope number. So if we want a SOUND to be controlled by envelope number 2 we would write the SOUND command as follows:

```
SOUND C,2,P,D
```

If envelope numbers above 4 are used, this may conflict with the operation of the filing system, so if files are being used it may be wise to limit the envelope numbers to the range 1 to 4.

When an envelope is generated it is divided up into a series of short time intervals or steps. The second parameter T sets up the length of each step time of the envelope and is measured in hundredths of a second. The value of T can range from 0 to 127

although there seems little point in setting T = 0. Generally the value for T will be 1 or 2 for most purposes.

The next six parameters are concerned with controlling variations in pitch as the sound is produced. The first three of these (PI1,PI2 and PI3) determine the rate at which pitch will change during each step time of the envelope. The other three terms (PN1,PN2 and PN3) control the number of steps over which the pitch change occurs. This effectively controls the amount of pitch change produced. We shall look more closely at these terms later.

The remaining six terms are all concerned with controlling the amplitude variations and levels and we shall look at these more closely in a moment.

Attack and decay

The whole character of a sound is governed by the way in which it builds up in volume and then dies away. The build up of sound is called the *attack* and the process of dying away is called *decay*. The rate of build up and the maximum loudness as well as the rate at which the sound dies away can be controlled by the last six terms in the ENVELOPE command.

Basically the amplitude envelope is divided into four phases, as shown in Fig. 9.3. The first is the attack phase and in the

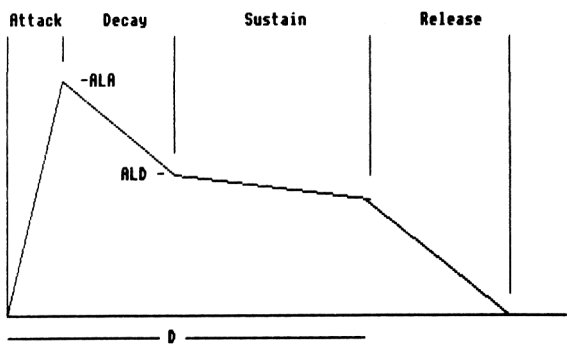


Fig. 9.3. The amplitude envelope.

ENVELOPE command two terms (AA and ALA) control the attack characteristics of the sound produced. The term AA is the rate of change of amplitude per step. Note that the amplitude will change at each step whose length is determined by the term T measured in hundredths of a second. For most purposes we will have

set T to 1 so each envelope step is 1/100 of a second. The second attack term is the attack amplitude limit level ALA which can have values from 0 to 127. Whereas the SOUND command allowed us only 16 steps of loudness the ENVELOPE command permits much finer control with its 128 steps.

The values of AA and ALA will actually determine the length of the attack phase. Suppose we set AA at 2 and ALA at 100, then since each step produces an increase of 2 it will take 50 steps to reach the ALA level assuming that the amplitude starts off at 0. If T is set at 1 then each step is 1/100 second so the length of the attack phase will be ½ second. The larger we make AA the faster the volume will rise and the larger we make ALA the louder the sound will be.

The attack step size can be in the range −127 to +127, with negative numbers indicating a fall in amplitude. This may seem a bit silly but in fact can be useful if the new sound interrupts a previous sound which ended with the amplitude still high. For most purposes, however, the value of AA will be positive. There is little point in using a value of 0 for AA unless you can be certain that a previous sound will always end with some useful amplitude level.

After the attack phase is completed the operation moves into the second phase of the ENVELOPE control, the decay phase. Here the amplitude will usually fall away. As for the attack phase there is a decay step size AD and a decay limit value ALD. Again the length of the decay phase will be determined by the rate of decay per step and the number of steps needed for the amplitude to reach the level ALD.

As with the attack stage the value of AD can be from −127 to +127 but would normally be a negative number. Making AD zero is not recommended since the phase will tend to go on forever unless ALD and ALA are equal. A positive value for AD could be used to make the sound increase perhaps at a different rate from that in the attack phase. In this case ALD must be greater than ALA. Normally ALD will be less than ALA.

Following the decay phase there is a third stage in the ENVELOPE generation called the *sustain phase.* Here there is only a step size term, which is between 0 and −127 in value. No limit value is specified but this phase will end when the amplitude has fallen to zero or alternatively when the duration time specified in the SOUND command has run out. This allows the decay to have two different rates if desired, or it may act as the decay control if the decay phase were used to increase the amplitude.

Finally there is what is called the *release* phase which occurs when

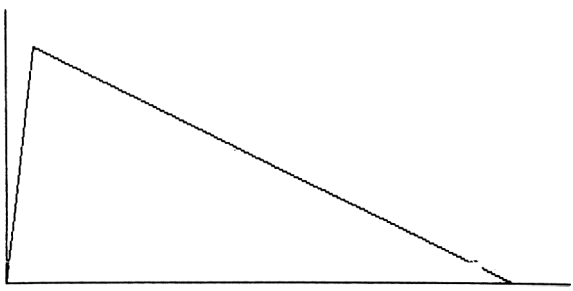
the note duration specified by the SOUND command runs out and this phase allows any remaining amplitude to die away to zero. If there is another SOUND command for the same channel waiting to be executed then the sound is cut off immediately to allow the next sound to be produced. The release phase may continue after the end of the duration period of the SOUND command. As with sustain there is only one term (AR) which determines the step size and it can have values from 0 to −127. This phase can be useful in terminating notes that are decaying very slowly or it can be used as a further rate of decay for a sound after the normal decay and sustain phases. If AR is set at 0 then any sound being produced at the end of the sustain period will continue indefinitely unless a new SOUND command is issued for that sound channel.

Now all of this may seem a little complicated at first, so let us look at a few examples of the ways in which these various ENVELOPE phases can be used.

Bells and chimes

Let us start by considering the sound produced by a bell or chime. The actual tone produced is simply a constant note, but the thing which gives the bell its characteristic sound is the way that the loudness builds up and dies away. In other words the attack and decay of the sound.

To produce sound from a bell it is struck by some kind of hammer. The impact of the hammer causes the bell to vibrate violently and the sound amplitude rises very rapidly to its maximum level. After being struck the bell continues to ring and the sound slowly dies away until all of the energy of the hammer blow has been used up. The resultant sound envelope is as shown in Fig. 9.4.



```
ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,-1,126,0
```

Fig. 9.4. Envelope for bell and chime type sounds.

We can emulate a bell sound by arranging that the attack phase of the envelope is very short and the decay is very slow. This is done by making AA=126 and ALA also 126 so that the attack phase is just one $\frac{1}{100}$ second step. For the decay phase we might use AD at −1 and ALD at 0, giving about 1¼ second decay time. The sustain and release terms might also be set at −1 and in the SOUND term we might set a duration of say 2 seconds or D=40.

Try the following program and you will see that quite a realistic bell type sound is produced:

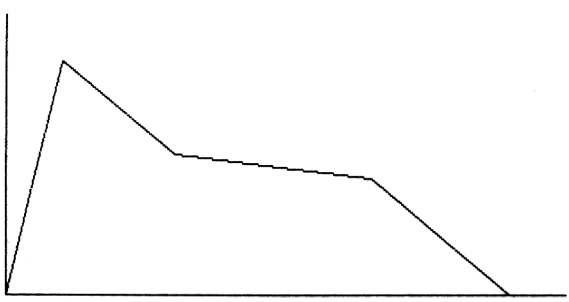
```
100 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,
    -1,-1,126,0
110 FOR N=53 TO 113 step 4
120 SOUND 1,1,N,40
130 NEXT N
```

Drums and other instruments

If instead of using a tone channel we had used the noise channel with a pitch value of 4 and the amplitude envelope of a bell type sound we would get a cymbal like sound. Better results are obtained if the decay rate is increased to about −2 or −3.

If the decay rate is increased to about −10 the effect becomes more like a side drum. By using pitch number 5 or 6 other different drum like sounds can be produced. Explosion noises are produced in a similar way to drum sounds.

If a normal tone channel is used with the rapid attack and with a decay rate of about −5 the effect becomes similar to that of a plucked instrument. This noise can also be effective as a bat meets ball sound for games.

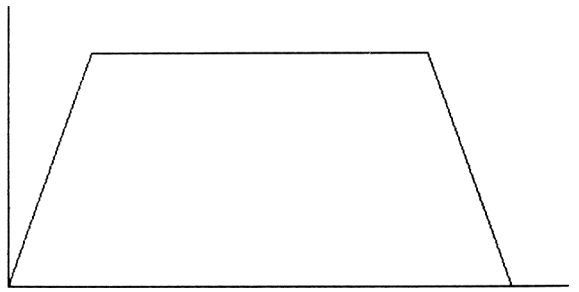


ENVELOPE 1,1,1,-1,0,1,1,200,126,-2,-1,-20,126,60

Fig. 9.5. Envelope for piano type sounds.

For a piano type sound the attack is made very fast but a slow decay to an amplitude level of perhaps 60 is used followed by a zero change in the sustain period and a fairly rapid decay in the release phase. This is shown in Fig. 9.5.

For organ and other wind type instruments the attack is rather slow and decay and sustain may keep the amplitude more or less level followed by a rapid decay in the release period, as shown in Fig. 9.6.



```
ENVELOPE 1,1,0,0,0,0,0,0,50,0,0,-50,126,126
```

Fig. 9.6. Envelope for organ type sounds.

Fig. 9.7 shows a selection of envelope commands for various types of instrument sound.

Setting up the pitch terms

Normally when we use the SOUND instruction alone, the pitch of the note produced remains constant throughout the duration of the sound. This is of course perfectly adequate for some purposes, such as producing a beep, but in many cases we will want to be able to vary the pitch of the note as it is being produced. This could be done by having a succession of sound commands with different pitch values, but a better solution is to use the ENVELOPE command to control the pitch as the note is generated.

For pitch control the ENVELOPE allows three separate periods of time during which the pitch can be varied. These three segments follow one another and the length of each segment is determined by the value of PN1, PN2 or PN3. Each of these numbers gives the length of that envelope segment in hundredths of a second and each may be a number from 0 to 255.

Fig. 9.8 shows the general idea of the pitch envelope. Here the pitch rises during the first section, then falls slowly during

Side drum

```
100 ENVELOPE1,1,0,0,0,0,0,0,126,-10,0,0
    ,126,0
110 FOR N=1 TO 30
120 SOUND0,1,4,10
130 SOUND0,1,4,5
140 SOUND0,1,4,15
150 NEXT N
160 END
```

Drum and cymbal

```
100 ENVELOPE1,1,0,0,0,0,0,0,60,-2,0,0
    ,126,0
110 ENVELOPE1,1,0,0,0,0,0,0,126,-4,0,0
    ,126,0
120 FOR N=1 TO 30
130 SOUND0,2,6,5
140 SOUND0,2,6,10
150 SOUND0,1,4,10
160 SOUND0,2,6,15
170 NEXT N
180 END
```

Vibraphone

```
100 ENVELOPE1,1,1,-1,1,1,2,1,30,0,0,
    -30,126,126
110 FOR N=100 TO 180 STEP 4
120 SOUND1,1,N,10
130 SOUND1,0,0,2
140 NEXT N
150 END
```

Woodwind

```
100 ENVELOPE1,1,1,-1,0,1,1,0,40,0,0,-40
    ,120,120
110 FOR N=1 TO 10
120 READ P
130 SOUND1,1,P,10
140 SOUND1,0,0,2
150 NEXT N
160 DATA 100,108,116,120,128
170 DATA 136,144,148,156,164
180 END
```

Fig. 9.7. Selection of envelope commands for various types of instrument sound.

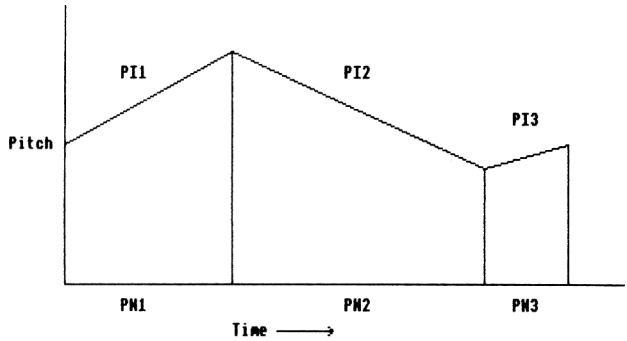


Fig. 9.8. The pitch envelope.

the second section and finally rises again during the third section.

The rate at which the pitch changes during each section of the envelope is governed by the numbers PI1, PI2, and PI3. These numbers give the change of pitch at each step in the section of the pitch envelope. Since the steps are always $1/100$ second then it is the rate of change of pitch per hundredth of a second. If the pitch is required to rise the PI number will be positive. Negative numbers of PI causes the pitch to fall during that segment of the envelope. So in our envelope the PI1 value will be positive, PI2 will be negative and P13 will be positive.

Normally the pitch changes will follow the three stages set up by

```
100 REM Siren type sounds
110 FOR J=1 TO 5
120 FOR K=2 TO 10 STEP 2
130 ENVELOPE 1,1,1,-J,1,K*J/2,K,K*J/2,
    126,0,0,-100,126,126
140 SOUND1,1,125,80
150 SOUND 1,0,0,10
160 NEXT K
170 NEXT J
180 FOR J=1 TO 5
190 FOR K=2 TO 10
200 ENVELOPE 1,1,J,-J,0,K,K,K,26,0,0,
    -100,126,126
210 SOUND1,1,125,80
220 SOUND1,0,0,10
230 NEXT K
240 NEXT J
250 END
```

Fig. 9.9. Program to demonstrate siren sounds.

the envelope and then the pitch sequence repeats for the rest of the sound duration, assuming that the duration is longer than the time specified by the pitch envelope. For some sounds, however, we may want the pitch change pattern to occur once only and this can be achieved by adding 128 to the value of the T term in the ENVELOPE statement. The pitch then remains at its last value until the duration specified in the SOUND statement is complete.

Using pitch variations

One obvious possibility of the pitch control is to produce various kinds of siren type sounds. Try out the program listed in Fig. 9.9 to see how this works. In all of these only the first pair of PI and PN terms are used. The pitch step is alternately 1 or −1 and the number of steps is large to give a large pitch change in most of these effects.

By using a +1,−1 pitch change per step and perhaps 1 to 5 steps for each pitch change, a vibrato effect is produced. This is shown by the short program of Fig. 9.10. Here again it is interesting to experiment

```
100 REM Vibrato type sounds
110 FOR J=0 TO 3
120 FOR K=2 TO 4 STEP 2
130 ENVELOPE 1,1,1,-J,1,K*J/2,K,K*J/2,
    126,0,0,-100,126,126
140 SOUND1,1,125,40
150 SOUND 1,0,0,10
160 NEXT K
170 NEXT J
180 FOR J=0 TO 3
190 FOR K=1 TO 4
200 ENVELOPE 1,1,J,-J,0,K,K,K,26,0,0,
    -100,126,126
210 SOUND1,1,125,40
220 SOUND1,0,0,10
230 NEXT K
240 NEXT J
250 END
```

Fig. 9.10. Demonstration of vibrato type effects.

with the size and number of pitch steps. Note that pitch changes of +1,−1 can be applied to the noise channel if a pitch number of 1 or 5 is used, and this produces interesting results.

For the sound of a bomb dropping, which might be useful in an arcade type game, the pitch starts high and then runs down over a

wide range. Here only one of the pitch envelope segments need be used, as shown in the program of Fig. 9.11. By using a similar pitch envelope with a sharp attack and fairly fast decay of amplitude, a chirping type of sound can be produced.

The possibilities of using the amplitude and pitch controls provided by the ENVELOPE command are almost limitless and the best way to explore the possibilities is just to experiment with a range of different envelopes and make a note of the parameters when an interesting sound is produced.

```
100 REM Bomb drop, explosion and laser
110 ENVELOPE1,132,-1,0,0,50,0,0,20,-1,
    -1,-12,126,0
120 ENVELOPE2,1,0,0,0,0,0,0,126,-1,0,
    -20,126,30
130 FOR N=1 TO 8
140 SOUND 1,1,150,30
150 SOUND 0,0,0,30
160 SOUND 1,0,0,20
170 SOUND 0,2,6,20
180 FOR T=1 TO 5000:NEXT T
190 NEXT N
200 ENVELOPE3,129,-2,0,0,40.0,0,126,-1,
    -1,-1,126,0
210 FOR N=1 TO 10
220 SOUND 1,3,150,20
230 FOR T=1 TO 1000:NEXT T
240 NEXT N
250 END
```

Fig. 9.11. Demonstration of bomb drop and explosion noises.

Chapter Ten
Making Music

Now that we can produce tones of selected frequency or pitch and of any desired duration, it becomes possible to turn the BBC Micro into a musical instrument. It can either be arranged to play some predetermined tune or alternatively we can arrange that the computer keyboard acts as the keyboard of our instrument so that it can be played in much the same way as a piano might be played. First, however, we need to take a brief look at some of the principles of music.

In music, the *tones*, or notes as they are usually called, have a definite relationship to one another. The complete pitch range is divided up into a series of octaves and each actave contains a set of 12 notes. As we move up in pitch the notes in one octave have approximately twice the frequency of the corresponding notes in the next lower octave.

The ratio of pitch between two successive notes in an octave is constant and equal to the twelfth root of 2. The pitch change between successive notes is called a *semitone*. If we examine an octave of notes on, say, a piano as illustrated in Fig. 10.1 it consists

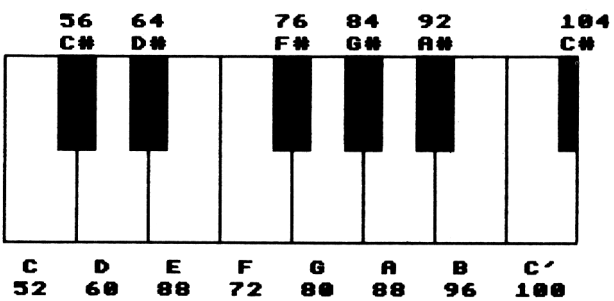


Fig. 10.1. Relationship between notes on a piano and pitch numbers for the SOUND command over one octave above middle C.

of seven *natural* notes which are produced by the white piano keys. The natural notes are labelled A, B, C, D, E, F and G in order of ascending pitch.

Between these natural notes there are a set of *sharp* notes which are a semitone higher in pitch than the adjacent natural note. The sharp notes use the same letters as natural notes but the letter is followed by a # symbol. Thus the note C# is a semitone higher than C. Note, however, that there are only five sharp notes. Most of the natural notes are separated from the next natural note by a sharp note so the pitch difference between adjacent natural notes is two semitones or a whole tone. The exceptions are B and E, which have no corresponding sharp note so the pitch difference between B and C or between E and F is only a semitone. This rather odd arrangement

Note name	Frequency Hertz	Pitch number
C	262	52
C#	277	56
D	294	60
D#	311	64
E	330	68
F	349	72
F#	370	76
G	392	80
G#	415	84
A	440	88
A#	466	92
B	494	96
C	523	100
C#	554	104
D	587	108
D#	622	112
E	659	116
F	698	120
F#	740	124
G	784	128
G#	831	132
A	880	136
A#	932	140
B	988	144
C	1047	148

NOTE frequencies in Hertz given as nearest integer number.

Fig. 10.2. Musical notes with their frequencies and pitch numbers.

is called the *chromatic* scale and if the natural notes are played in succession we get the familiar musical scale. Try running the following program which will generate the scale for the octave starting at middle C.

```
100 DIM P%(7)
110 P%(1)=52:P%(2)=60:P%(3)=68:P%(4)
    =72
120 P%(5)=80:P%(6)=88:P%(7)=96:P%(8)
    =100
130 FOR I=1 TO 10
140 FOR N=1 TO 8
150 SOUND 1,-15,P%(N),20
160 SOUND 1,0,0,0
170 NEXT N
180 NEXT I
190 END
```

In Fig. 10.2 a listing of the note names, frequency in hertz and the required pitch number for the SOUND command are given for the two octaves above middle C. In music you will also find references to *flat* notes which are a semitone below a natural note. Sharps and flats are just different ways of labelling the same note so that A sharp, which is a semitone above A, is the same as B flat, which is a semitone below B. In written music the flat is denoted by a symbol like a small b placed after the note, whilst a sharp is denoted by the crosshatch symbol (#) after the note.

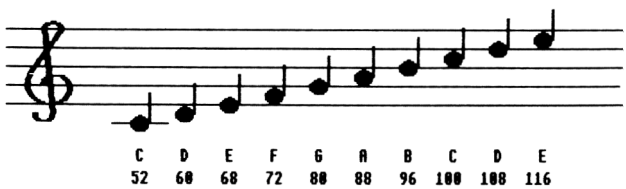


Fig. 10.3. The treble stave in music showing relations between pitch numbers and musical notes.

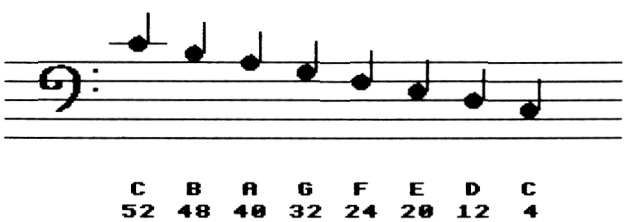


Fig. 10.4. The bass stave of music and related pitch values for notes. Note that middle C is on the line above the stave, and is equal to the C below the line on the treble stave.

When notes are written on paper they are usually shown as large dots with vertical tails drawn on or between a series of horizontal lines called the *stave*, and will appear as shown in Fig. 10.3. This diagram shows the treble stave and indicates the pitch numbers for each of the notes. A second stave for lower notes is called the bass stave and is shown in Fig. 10.4.

It will be noted that for each semitone increase in pitch the pitch number increases by four. Thus most natural notes are eight counts apart except that B,C and E,F, and the sharp notes are four higher than their corresponding natural notes. Thus moving to a note a complete octave higher adds a count of 48 to the pitch value.

Musical timing

Apart from the pitch, music also makes use of variable duration of the notes, organised on a binary system. The basic note length is called a *crotchet* and corresponds to a duration of about 10 or roughly half a second. The crotchet is shown as a black filled circle with a tail. Shorter notes are the *quaver* (½), *semiquaver* (¼) and *demisemiquaver* (⅛) which are drawn like crotchets but have one, two or three ticks on the tail respectively. A longer note is the *minim* which is twice the length of a crotchet and is shown like a crotchet but with the circle not filled in. Finally, there is the *semibreve* which is four times as long as a crotchet and is shown with no tail. These note durations are summarised in Fig. 10.5.

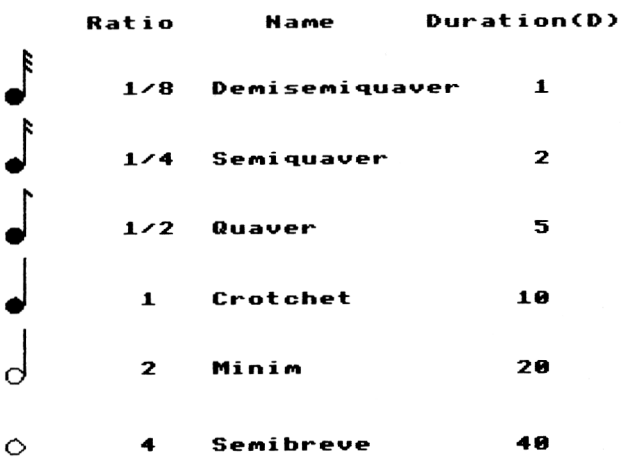


Fig. 10.5. Duration of musical notes showing the symbol used, relative timing and duration number.

Sometimes in written music you may also come across notes with a dot alongside the note symbol. These notes have the duration increased by half. Thus a dotted crotchet would have a duration value of 15 instead of the normal 10.

To make the computer play a tune we can simply set up a data statement with a pair of numbers, one for pitch and one for duration, for each note of the tune to be played. To make the computer play the music, the program simply takes successive values for pitch and duration and then inserts these into the SOUND statement. This playing sequence may be a simple FOR-NEXT loop which runs through all of the notes in the tune. Try running the program listed in Fig. 10.6 to see how this works.

```
100 REM Tune playing program
110 DIM P%(32),D%(32)
120 FOR N=1 TO 26
125 READ P%(N),D%(N)
130 NEXT N
140 O%=0
150 FOR K=1 TO 3
160 FOR N=1 TO 26
170 SOUND 1,-15,P%(N)+O%,D%(N)
180 SOUND 1,0,0,0
190 NEXT N
200 SOUND 1,0,0,20
210 O%=O%+48
220 NEXT K
230 DATA 96,10,108,5,96,10,96,10
240 DATA 108,5,96,10,100,5,108,5
250 DATA 100,5,88,10,96,5,100,5
260 DATA 96,5,80,10,96,10,108,5
270 DATA 96,10,96,10,108,5,96,10
280 DATA 100,5,108,5,100,5,88,10
290 DATA 96,5,80,10
300 END
```

Fig. 10.6. Program to play a simple musical tune.

Making richer sounds

So far we have tended to use only one sound at a time, but musicians discovered many centuries ago that by playing two or more different notes at the same time a much richer and more harmonious sound could be produced. Not all combinations of notes are successful in this respect. Some combinations of notes harmonise together to

produce a pleasant sound but other combinations will produce a discordant and harsh result. Understanding the best combinations really requires some knowledge of musical principles but for our purposes we can perhaps suggest some possible combinations. First, we can use notes spaced one octave apart so that the C of one octave is combined with the C of the next octave above. The higher note has roughly twice the frequency of the lower note and is therefore the second harmonic. In the BBC Micro this is easily done since the higher of the two notes will always have a pitch value 48 higher than that of the lower note. Other combinations with notes two or more tones above the basic note will also produce interesting results. To see the effect of adding these notes, try running the following program. This alternately plays a set of notes alone and with a second note added.

```
100 FOR K% = 16 TO 48 STEP 8
110 FOR J = 1 TO 2
120 FOR P% = 100 TO 148
130 A=-15
140 SOUND 1,-15,P%,10
150 IF J=1 THEN A=0
160 SOUND 2,A,P%+K%,10
170 SOUND 1,0,0,2
180 SOUND 2,0,0,2
190 NEXT P%
200 FOR T=1 TO 1000:NEXT T
210 NEXT J
220 NEXT K%
230 END
```

Since there are three sound channels, you could modify the program to add a third note to the combination. It is important, however, to ensure that SOUND commands for all three channels are used in each pass through the loop and that the unwanted notes are suppressed by making their amplitude zero rather than leaving out the SOUND command. We shall see the reason for this later.

Queueing and flushing

When SOUND commands are processed by the BASIC interpreter in the BBC Micro the computer does not wait for each sound to complete before going on to the next instruction. Try running the following short program.

```
100 REM SOUND QUEUE
110 CLS
120 SOUND 1,-15,110,100
130 PRINT"NOTE 1"
140 SOUND 1,-15,120,100
150 PRINT"NOTE 2"
160 SOUND 1,-15,130,100
170 PRINT"NOTE 3"
180 SOUND 1,-15,140,100
190 PRINT"NOTE 4"
200 SOUND 1,-15,150,100
210 PRINT"NOTE 5"
220 SOUND 1,-15,160,100
230 PRINT"NOTE 6"
240 SOUND 1,-15,170,100
250 PRINT"NOTE 7"
260 SOUND 1,-15,180,100
270 PRINT"NOTE 8"
280 SOUND 1,-15,190,100
290 PRINT"NOTE 9"
300 SOUND 1,-15,200,100
310 PRINT"NOTE 10"
320 END
```

When this program is run you might expect that the messages "NOTE 1", "NOTE 2", etc., would be printed as each note is produced. In fact you will see that the first six PRINT statements are executed almost immediately whilst the first sound is being generated. When the first note ends, the PRINT statement for note 7 is printed and then a new PRINT occurs as each sound completes.

When the first SOUND command occurs it is executed immediately and the program goes on to process the following PRINT command. When the second SOUND request for channel 1 is reached, the sound generator is busy producing the first note. To avoid holding up the program this new SOUND command is accepted and placed in a queue in memory and program execution continues with the following PRINT statement. Further SOUND commands may be added to the queue as they occur. In the BBC Micro the queue for each sound channel can hold up to four SOUND commands. Whilst the queue is filling, the computer continues to execute its BASIC program and just pushes SOUND commands on to the queue. When the queue is full the computer will wait at the next SOUND command. When the first sound completes, the next in the queue is executed automatically. The commands in the queue now shuffle up and a new command can be entered at the end of the queue so the computer can move a little

further through its program. In the program you will see that when the first note ends, the message "NOTE 7" is printed indicating that a new SOUND command has been accepted and placed in the queue. Similarly the messages for notes 8, 9 and 10 will appear as earlier notes in the queue are completed.

One problem with this queuing scheme is that if you want to produce a sound in response to some event, such as pressing a key on the keyboard, the queue scheme is inconvenient. If there are already some SOUND commands in the queue then the new sound will not be produced until these have been dealt with. There is, however, a way to overcome this problem. The BBC Micro provides a facility which allows any commands already in the queue for a sound channel to be cleared out or *flushed* so that the new SOUND command is executed immediately.

Earlier we mentioned that the channel number parameter did more than just select a channel. In fact the parameter C is a four-digit hexadecimal number with digits H, S, F and N. We have already used the last, or least significant, digit N, since this is the channel number. The other three digits are:

```
SOUND &HSFN
```

Here the F digit is used to flush the sound queue. Under normal conditions where F=0 nothing happens to the orders in the queue and they are processed in sequence. If F=1 then all existing SOUND orders in the queue are cancelled and the new order is executed next.

To see how this works change the SOUND statement at line 26Ø to the following:

```
260 SOUND &11,-15,180,100
```

Now when the program is run the first note sounds and the first six messages are printed. After the first note has ended, the messages for notes 7,8,9 and 10 will appear and the sound for note 8 is generated followed by notes 9 and 10. In this case notes 2 to 7 were flushed out of the sound queue when the statement for note 8 was read by the computer.

When the flush order or the digits H or S of the C parameter in the SOUND command are used, the C value is preceded by the & sign to indicate that the number is hexadecimal.

This technique of flushing the sound queue is useful when we make use of the computer keyboard as a musical keyboard. As soon as a new key is pressed for a different note then any notes that may be in the sound queue are flushed out so that the new note starts

immediately. Another application for flushing is in games, where there may be some more or less continuous background sound but when a hit is scored or some other event occurs we want to interrupt the sound and introduce some new sound such as an explosion or perhaps a victory or defeat tune.

Sometimes when flushing the sound queue we may want to allow the current sound to complete before starting the new note. This is where the H digit of the C parameter comes into play. Normally H is 0 and if the queue is flushed the current sound being generated is also terminated. If H is set at 1 then the current sound will go on to completion. Note that if H is set at 1 then all four digits of the C term must be included, so that if we want to flush channel 1 and complete the current sound the command becomes

```
100 SOUND &1011,A,P,D
```

Chords and synchronisation

We have seen that combining two tones together produces a richer sound. This technique is used regularly in music where the combination of two or three notes played together is called a *chord*. To play a chord the two or three notes of the chord are generated at the same time by producing each note from a different channel of the sound generator.

The sound queue system can cause problems when we are playing chords, since each sound channel has its own queue and these operate independently of one another. All would be well if, say, we were playing pairs of notes and there were the same number of notes from each channel, since the queues of notes for the two channels will be equal in length and the pairs of notes will move along the two queues in step. In music, however, we find a mixture of chords and single notes following one another. In this case one channel may have a different number of notes in its queue. When a single note is played from, say, channel 1 there is no corresponding note in the channel 2 queue, but the sound generator will simply produce the next note in that queue which will in fact be one of the notes for the next chord. At this point the two channels will get out of step and within a very short time chaos will reign. Obviously we need some scheme that will keep the two queues of notes in step, even when there are single notes between the chord notes in one queue.

One possible approach to this problem would be to insert dummy

notes into the second queue. These dummy notes would have zero amplitude and the same duration as the corresponding note in the first queue. Now the two queues will step along together and the chord notes will be played together as required.

The alternative technique for keeping the queues in step is to use some method of marking the notes in the two queues that have to be played together. Now whilst queue 1 plays any single notes in the sequence the second queue waits. This can be achieved on the BBC Micro by using the synchronisation (S) digit in the channel code number of the SOUND command. In this case the S digit acts as a label and effectively marks those notes in the channel 1 queue which have to be played with a note from the channel 2 queue. In fact the common notes in both queues are marked with an S digit of 1. Notes to be played singly in the channel 1 queue are marked with S = 0. Now as the notes are played, if the S value in channel 1 is 0 the channel 1 note is played but the channel 2 queue is held in position and no sound is produced on channel 2. When an S value of 1 occurs in channel 1 the note will not be produced until a corresponding note with S = 1 is available for channel 2. Looking at this a different way, the channel 2's sound queue will wait on the channel 1 queue and will only produce a note when there is a note marked with S = 1 in the channel 1 queue.

The synchronisation process can be extended to cover three or even four notes by setting the S value of the notes to be played together to 2 or 3. The S digit will in fact tell the sound generator how many other notes with that same flag number it has to wait for before the chord is sounded.

Making a keyboard instrument

To turn the computer into a playable instrument we can allocate keys on the keyboard to musical notes and the logical arrangement is to select a layout similar to that on a piano. In the program listed in Fig. 10.7 the keys Q to @ are used for the natural notes and correspond to the white keys on a piano. The number keys 2,3,5,6,7,9 and ∅ are used for the sharp notes and correspond to the black piano keys. The relative positions of the black and white keys is the same as for a piano keyboard.

By using INKEY with a negative argument it is possible to find out if a particular key on the keyboard is being pressed. This command, unlike the normal INKEY function, actually reads the

```
LIST
  100 REM Musical keyboard instrument
  110 DIM K%(32),F%(32)
  120 MODE1
  130 PROCDISPL
  140 FOR N=1 TO 32
  150 READ K%(N)
  160 F%(N)=0
  170 NEXT N
  180 O%=48
  190 E%=1
  200 ENVELOPE 1,1,0,0,0,0,0,0,126,-1,0,
-1,126,0
  210 ENVELOPE 2,1,0,0,0,0,0,0,126,-1,-1
,-5,126,80
  220 ENVELOPE 3,1,1,-1,0,1,1,0,20,0,0,-
20,126,126
  230 ENVELOPE 4,1,0,0,0,0,0,0,50,-1,0,-
10,120,90
  300 REPEAT:FOR N=1 TO 32
  310 IF INKEY(-K%(N))=F%(N) THEN PROCSN
D
  320 NEXT N:UNTIL 0
  340 END
  500 DEFPROCSND
  510 IF F%(N)=0 SOUND &1011,0,0,0:F%(N)
=-1:GOTO 560
  520 IF N>29 THEN O%=48+48*(N-30)
  530 IF N>22 AND N<27 THEN E%=N-22
  540 IF N<23 THEN SOUND 1,E%,O%+4*N,-1

  550 F%(N)=0
  560 ENDPROC
  600 DEFPROCDISPL
  610 MOVE 120,500
  620 PLOT1,1040,0
  630 PLOT0,0,300
  640 PLOT1,-1040,0
  650 VDU5
  660 FOR J=1 TO 13
  670 PLOT1,0,-300
  680 READ N$,S$
  690 PLOT0,24,-16
  700 PRINT N$;
  710 PLOT0,-56,316
  720 IF S$=" " THEN 800
  730 PLOT0,-16,48
  740 PRINT S$;
  750 PLOT0,-36,-48
  760 PLOT0,0,-100
```

```
770 PLOT81,40,100
780 PLOT81,0,-100
790 PLOT0,-20,100
800 PLOT0,80,0
810 NEXT J
820 PLOT1,0,-300
830 VDU4
840 PRINT TAB(0,20)"f8=Normal"
850 PRINT"f9=One octave higher"
860 PRINT
870 PRINT"f0=Bell":PRINT"f1=Piano"
880 PRINT"f2=Woodwind":PRINT"f4=Organ"
890 PRINT"f5=Electronic"
900 PRINT:PRINT"PLAY NOW"
910 ENDPROC
920 DATA "Q"," ","W","2","E","3","R"
930 DATA " ","T","5","Y","6","U","7"
940 DATA "I"," ","O","9","P","0","@"
950 DATA " ","[","^","_","\"
960 DATA 17,50,34,18,35,52,20,36
970 DATA 53,69,37,54,38,39,55,40
980 DATA 56,72,25,57,121,41,33,114
990 DATA 115,116,21,117,118,23,119,120
```

Fig. 10.7. Program to convert the BBC Micro into a simple keyboard instrument.

state of the keys on the computers keyboard. To find out which key is being pressed, each of the keys used is tested in turn by INKEY (-K%(N)) in a loop. When the key is being pressed the result of the test is −1 but otherwise the result is 0. The key codes K% are initially read in as an array using READ and DATA statements and a READ loop. A second array F% is also set up and this is used to store the opposite of the last state of each key. The INKEY result for each key is compared with the state of its flag F%(N) to see if there is any change. If not, the loop goes on to the next key. Thus at the start all F% values are set to 0 which is the state from INKEY when the key is pressed. If the key is pressed then its F% is changed to −1 so that following INKEY commands, check to see if that key has been released.

When a change of key state occurs the program goes to a procedure which interprets the key action and sets up the sound generator operation. On entry to the procedure the value of the loop count N indicates which key has changed state. If the current state F%(N) of the key is 0 then the key has just been pressed and a new note must be started. The pitch of the new note is calculated by simply multiplying N by 4 and adding this to a fixed number

representing a note one semitone lower than the first note on the keyboard. In this case the reference value is 48 which corresponds to the B below middle C. By setting the keys up in the correct sequence in the test loop, all notes will now have the required pitch. The note duration is set at −1 so the note will sound until it is cancelled by a new SOUND command.

Function keys f8 and f9 are used to select two alternative octaves for the keyboard. Function keys f0 to f5 are used to select different envelopes for the sound and will give the keyboard a selection of different voices, such as bells, organ, piano and a vibrato effect.

Index

actual colours, 72
aircraft landing program, 134
all round viewing, 130
amplitude envelope, 147
amplitude of sounds, 141
animated man, 111
animation, 104
arcs, drawing of, 24
arrow keys, use of, 13
attack phase, 147

background colour, 44, 46, 92
background colour numbers, 46
bar charts, 55, 116
bell type sounds, 149
bomb dropping sound, 155
bouncing ball, 106

chime sounds, 149
circles, drawing of, 23, 26, 28, 34
circular plots, 123
circular scales, 65
CLG command, 45
clock displays, 64
CLS command, 45
collisions, detection of, 108
colour bit allocation, 72
COLOUR command, 39
colour control codes, 91
colour fill, 46
colour, inverted, 76, 81
colour, logical, 72
colour mixing, 83
colour numbers, 72
colour palette, 71
colour switching, 74
colours, normal set of, 41
contiguous graphics, 98
continuation of sounds, 164
crotchet, 159
cymbal sound, 150

decay phase of sounds, 147
dials, drawing, 64
diamonds, drawing of, 33
display memory, 8
display modes, 6, 7
dot matrix for text, 5
dotted lines, 82
double height symbols, 99
DRAW command, 12
drawing scales, 53
drawing solid bars, 120
drum type sounds, 150
duration of sounds, 143

engine sounds, 145
ENVELOPE command, 148
erasing objects, 79
explosion sounds, 150

filling with colour, 46
flashing symbols, 98
flat notes, 158
flushing of sounds, 161
foreground colour numbers, 41

GCOL command, 41, 76
graph axes, 62
graph drawing, 61
graphs scientific, 62
graphs three axis, 115
graphics colour, 41, 95
graphics coordinates, 10
graphics cursor, 10, 53
graphics hold, 100
graphics pixel, 5
graphics release, 101
graphics symbols, 4, 90, 95
graphics window, 50

hexagons, drawing, 19
hit detection in games, 108

interpolation, 62
inverted colours, 81

kaleidoscope, 49
keyboard instrument, 165

line drawing, 12, 82
linking the cursors, 53, 112
logical GCOL operations, 77

making new symbols, 109
memory requirements, 8
minim, 159
mirror image patterns, 48
moiré patterns, 43
mosaic graphics, 5, 90, 95
MOVE command, 11
moving ball, 105
moving sector display, 68
multiple bar charts, 57
music, chords in, 161, 164
musical octave, 156
musical scale, 158

natural notes, 157
new symbols, making, 109
noise channel, 144

organ sounds, 151

palette, 72
periodic noise, 144
perspective, 121
perspective equations, 125
perspective scaling, 126
piano sound, 151
pie charts, 59
pitch envelope, 153
pitch, setting up, 142
pitch values for music,157
pitch variations, 151, 154
pixel graphics, 5
PLOT command, 32, 46, 81, 84
plotting single dots, 64
polygons, drawing of, 21, 30, 34
projection of 3D views, 126, 132

quadratic equation method, 26
quaver, 159

queueing of sound commands, 161

rectangle drawing, 18
relative coordinates, 32
release phase of sounds, 148
ribbon patterns, 42
rotation equations, 28, 36
rotation of figures, 36
rubber band drawing, 86

scales, drawing graph, 52
scaling, 34
scrolling of text, 113
semibreve, 159
semitones, 156
separated graphics, 98
sharp notes, 157
siren sounds, 153
sketching program, 13
SOUND command, 140
sound generator, 139
sound queue, 161
special symbols, creation of, 109
squares, drawing, 15
steam engine effect, 144
stretching, 34
surface maps, 123
sustain phase of sounds, 148
symbol space, 4
synchronisation of sound, 164

TAB function, 9
teletext background, 92
teletext mode, 89
text colour, 39, 91
text cursor, 9, 53
text window, 50
text with graphics, 53
thermometer display, 52
three axis graphs, 115
translation of points, 131

varying noise pitch, 145
VDU19 command, 73
VDU23 command, 110
vibrato effects, 154
video display, 3

white noise, 144
windows, 49
wire frame models, 130

THE ORIC-1 PROGRAMMER
M. James and S. M. Gee
0 246 12157 2

ORIC MACHINE CODE HANDBOOK
Paul Kaufman
0 246 12150 5

The TI 99/4A

GET MORE FROM THE TI99/4A
Garry Marshall
0 246 12281 1

The VIC 20

GET MORE FROM THE VIC 20
Owen Bishop
0 246 12148 3

THE VIC 20 GAMES BOOK
Owen Bishop
0 246 12187 4

The ZX Spectrum

THE ZX SPECTRUM And How To Get The Most From It
Ian Sinclair
0 246 12018 5

THE SPECTRUM PROGRAMMER
S. M. Gee
0 246 12025 8

THE SPECTRUM BOOK OF GAMES
M. James, S. M. Gee and K. Ewbank
0 246 12047 9

INTRODUCING SPECTRUM MACHINE CODE
Ian Sinclair
0 246 12082 7

SPECTRUM GRAPHICS AND SOUND
Steve Money
0 246 12192 0

THE ZX SPECTRUM How to Use and Program
Ian Sinclair
0 586 06104 5

Learning is Fun! 40 EDUCATIONAL GAMES FOR THE SPECTRUM
Vince Apps
0 246 12233 1

The ZX81

THE ZX81 How to Use and Program
S. M. Gee and Mike James
0 586 06105 3

Which Computer?

CHOOSING A MICROCOMPUTER
Francis Samish
0 246 12029 0

Languages

COMPUTER LANGUAGES AND THEIR USES
Garry Marshall
0 246 12022 3

EXPLORING FORTH
Owen Bishop
0 246 12188 2

Machine Code

Z-80 MACHINE CODE FOR HUMANS
Alan Tootill and David Barrow
0 246 12031 2

6502 MACHINE CODE FOR HUMANS
Alan Tootill and David Barrow
0 246 12076 2

Using Your Micro

COMPUTING FOR THE HOBBYIST AND SMALL BUSINESS
A. P. Stephenson
0 246 12023 1

DATABASES FOR FUN AND PROFIT
Nigel Freestone
0 246 12032 0

SIMPLE INTERFACING PROJECTS
Owen Bishop
0 246 12026 6

INSIDE YOUR COMPUTER
Ian Sinclair
0 246 12235 8

Programming

THE COMPLETE PROGRAMMER
Mike James
0 246 12015 0

PROGRAMMING WITH GRAPHICS
Garry Marshall
0 246 12021 5

Word Processing

CHOOSING A WORD PROCESSOR
Francis Samish
0 246 12347 8

WORD PROCESSING FOR BEGINNERS
Susan Curran
0 246 12353 2

Two of the most attractive features of the BBC Micro are its very versatile graphics and sound facilities.

This book provides a practical guide to the graphics and sound facilities, clearly setting out the techniques which allow you to make the most of the BBC machine's capabilities. You are shown the techniques of producing graphs and charts, drawing pictures and using colour. The book goes on to explain animation, an essential element of video games, as well as perspective and the world of 3D graphics. The operation of the sound generator is explored both for sound effects and musical applications.

Many short, easily handled, illustrative program listings are provided as well as several complete program listings for you to try out yourself.

The Author

Steve Money is a well-known author of several books including *Microprocessor Data Book* published by Granada.

Other books on the BBC Micro from Granada

INTRODUCING THE BBC MICRO
Ian Sinclair
0 246 12146 7

THE BBC MICRO
An Expert Guide
Mike James
0 246 12014 2

Learning is Fun!
40 EDUCATIONAL GAMES FOR THE BBC MICRO
Vince Apps
0 246 12317 6

BBC MICRO AND ELECTRON MACHINE CODE
An Expert Guide
A P Stephenson and D J Stephenson
0 246 12227 7

DISCOVERING BBC MICRO MACHINE CODE
A P Stephenson
0 246 12160 2

21 GAMES FOR THE BBC MICRO
Mike James, S M Gee and Kay Ewbank
0 246 12103 3

ADVANCED PROGRAMMING FOR THE BBC MICRO AND ELECTRON
Mike James and S M Gee
0 246 12158 0

DISK SYSTEMS FOR THE BBC MICRO
Ian Sinclair
0 246 12325 7

Photograph of the BBC Microcomputer by courtesy of Acorn Computer Ltd and Optimus Graphic Design Ltd.

GRANADA PUBLISHING
Printed in Great Britain 0 246 12156 4

£6.95 net

MONEY **BBC MICRO GRAPHICS AND SOUND** GRANADA